UNIVERSIDAD POLITÉCNICA DE MADRID
Escuela Universitaria de Ingeniería Técnica de Telecomunicación

RECONFIGURABLE VIDEO CODE (RVC)

TRABAJO FIN DE CARRERA

Autor:

Rafael Pons Castellanos

Tutor:

Fernando Pescador del Oso

Doctor Ingeniero de Telecomunicación

Junio 2014

E.T.S.I.S. TELECOMUNICACIÓN

PROYECTO FIN DE CARRERA
PLAN 2000

PROYECTO FIN DE CARRERA	
Título	Aplicación para automatizar la asignación de procesos RVC a los núcleos de un sistema multiprocesador
Autor	Rafael Pons Castellanos
Tutor	Fernando Pescador del Oso
Ponente	
Tribunal	
Presidente	Juana Gutiérrez Arriola
Secretario	Eduardo Juárez Martínez
Vocal	

El Secretario:

AGRADECIMIENTOS.

Este Proyecto Fin de Carrera sólo ha sido viable con el apoyo de muchas personas, tanto del ámbito personal como profesional. Por este motivo quiero darles las gracias a todos.

A Fernando Pescador, tutor del presente Proyecto Fin de Carrera, por su implicación, apoyo y especialmente por realizar revisiones y correcciones continuas de las diferentes versiones de esta memoria logrando que estuviese perfectamente completada. Quiero agradecerle la gran oportunidad que me ofreció a la hora de poder realizar este Proyecto e introducir la cabeza en un mundo como es RVC, que me ha parecido gratamente interesante.

A Miguel Chavarrías por su orientación, apoyo, paciencia, por transmitirme los conocimientos necesarios y de suma importancia para poder entender y desarrollar el Proyecto. Quiero mencionar y resaltar la gran labor que realiza en el laboratorio, ofreciendo su ayuda a los alumnos que realizan el PFC y las facilidades que me proporcionó en todo momento tanto a la hora de realizar la memoria como del desarrollo de mi programa.

A Carlos Carrillo le quiero agradecer de forma muy especial su gran apoyo y paciencia por todo el tiempo que me ha dedicado en todo momento, enseñándome, corrigiéndome y aportándome ideas y soluciones para los problemas que me han surgido tanto a lo largo de la carrera como del PFC y que sin su ayuda no hubiera conseguido sacar a delante.

A mis padres y abuela, por ayudarme moralmente en el transcurso de mis estudios y, de alguna forma, “presionarme a esforzarme” para conseguir mis objetivos.

A Estefanía Reca por estar a mi lado desde el instituto, realizar la carrera de Ingeniería Técnica de Telecomunicaciones codo con codo y por ayudarme en todo momento. Espero que nuestra aventura juntos continúe por mucho tiempo.

A los profesores del grupo de electrónica, por la ayuda prestada en todo momento y por transmitirme los conocimientos necesarios para poder llegar a finalizar esta carrera de ingeniería.

A mis compañeros de laboratorio, y en especial a Daniel por el gran ambiente de trabajo que se respiraba en el laboratorio y por la ayuda prestada en los comienzos del proyecto y su aportación de ideas.

A todos y cada uno de vosotros, GRACIAS!!

ÍNDICE GENERAL.

ÍNDICE DE FIGURAS.

ÍNDICE DE TABLAS.

RESUMEN.

El esquema actual que existe en el ámbito de la normalización y el diseño de nuevos estándares de codificación de vídeo se está convirtiendo en una tarea difícil de satisfacer la evolución y dinamismo de la comunidad de codificación de vídeo. El problema estaba centrado principalmente en poder explotar todas las características y similitudes entre los diferentes códecs y estándares de codificación. Esto ha obligado a tener que rediseñar algunas partes comunes a varios estándares de codificación.

Este problema originó la aparición de una nueva iniciativa de normalización dentro del comité ISO/IEC MPEG, llamado Reconfigurable Video Coding (RVC). Su principal idea era desarrollar un estándar de codificación de vídeo que actualizase e incrementase progresivamente una biblioteca de los componentes, aportando flexibilidad y la capacidad de tener un código reconfigurable mediante el uso de un nuevo lenguaje orientado a flujo de Actores/datos denominado CAL. Este lenguaje se usa para la especificación de la biblioteca estándar y para la creación de instancias del modelo del decodificador.

Más tarde, se desarrolló un nuevo estándar de codificación de vídeo denominado High Efficiency Video Coding (HEVC), que actualmente se encuentra en continuo proceso de actualización y desarrollo, que mejorase la eficiencia y compresión de la codificación de vídeo. Obviamente se ha desarrollado una visión de HEVC empleando la metodología de RVC.

En este PFC, se emplean diferentes implementaciones de estándares empleando RVC. Por ejemplo mediante los decodificadores Mpeg 4 Part 2 SP y Mpeg 4 Part 10 CBP y PHP así como del nuevo estándar de codificación HEVC, resaltando las características y utilidad de cada uno de ellos. En RVC los algoritmos se describen mediante una clase de actores que intercambian flujos de datos (*tokens*) para realizar diferentes acciones.

El objetivo de este proyecto es desarrollar un programa que, partiendo de los decodificadores anteriormente mencionados, una serie de secuencia de vídeo en diferentes formatos de compresión y una distribución estándar de los actores (para cada uno de los decodificadores), sea capaz de generar diferentes distribuciones de los actores del decodificador sobre uno o varios procesadores del sistema sobre el que se ejecuta, para conseguir la mayor eficiencia en la codificación del vídeo.

La finalidad del programa desarrollado en este proyecto es la de facilitar la realización de las distribuciones de los actores sobre los núcleos del sistema, y obtener las mejores configuraciones posibles de una manera automática y eficiente.

ABSTRACT.

The current scheme that exists in the field of standardization and the design of new video coding standards is becoming a difficult task to meet the evolving and dynamic community of video encoding. The problem was centered mainly in order to exploit all the features and similarities between different codecs and encoding standards. This has forced redesigning some parts common to several coding standards.

This problem led to the emergence of a new initiative for standardization within the ISO / IEC MPEG committee, called Reconfigurable Video Coding (RVC). His main idea was to develop a video coding standard and gradually incrementase to update a library of components, providing flexibility and the ability to have a reconfigurable code using a new flow -oriented language Actors / data called CAL. This language is used for the specification of the standard library and to the instantiation model decoder.

Later, a new video coding standard called High Efficiency Video Coding (HEVC), which currently is in continuous process of updating and development, which would improve the compression efficiency and video coding is developed. Obviously has developed a vision of using the methodology HEVC RVC.

In this PFC, different implementations using RVC standard are used. For example, using decoders MPEG 4 Part 2 SP and MPEG 4 Part 10 CBP and PHP and the new coding standard HEVC, highlighting the features and usefulness of each. In RVC, the algorithms are described by a class of actors that exchange streams of data (*tokens*) to perform different actions.

The objective of this project is to develop a program that, based on the aforementioned decoders, a series of video stream in different compression formats and a standard distribution of actors (for each of the decoders), is capable of generating different distributions decoder actors on one or more processors of the system on which it runs, to achieve greater efficiency in video coding.

The purpose of the program developed in this project is to facilitate the realization of the distributions of the actors on the cores of the system, and get the best possible settings automatically and efficiently.

CAPÍTULO 1. INTRODUCCIÓN Y OBJETIVOS

1.1 Introducción

El desarrollo de sistemas de codificación de vídeo es un proceso cada vez más complejo y costoso que requiere un tiempo de desarrollo elevado, lo que provoca que el desarrollo de nuevos productos a largo plazo sea elevado, así como su aparición en el mercado. Para facilitar este proceso de desarrollo, el grupo de trabajo MPEG está desarrollando un nuevo estándar que facilite este proceso.

Este desarrollo del nuevo estándar para la codificación digital de vídeo reconfigurable, RVC (Reconfigurable Video Coding), tiene como principal objetivo proporcionar un área de trabajo con mayor capacidad de desarrollo. También propone una forma de trabajo dinámica, en continua evolución, así como la implementación y la adopción de soluciones de codificación de los estándares de vídeo anteriores, con características de mayor flexibilidad y capacidad de reutilización.

RVC ha desarrollado dos estándares: Codec Configuration Representation (ISO / IEC 23.001-4) y Video Tool Library (ISO / IEC 23002-4) [**13**], terminados en 2009. La biblioteca de herramientas de vídeo (VTL) especifica un conjunto de Unidades Funcionales (UF), "bloques de construcción" para los decodificadores de vídeo, como transformaciones, compensadores de movimiento o decodificadores de entropía. La estructura de un decodificador de vídeo (compuesta de unidades funcionales específicas y sus interconexiones) está definida por un Lenguaje de Descripción de Decodificador (DDL) y el formato del flujo de bits codificado se define utilizando la Sintaxis de Lenguaje de Descripción de Bitstream (BSDL), ambos especificados en la representación del códec.

El resultado de realizar un diseño de un codificador o decodificador mediante RVC es la obtención de un código fuente (en diferentes lenguajes de programación como C/C++, Java, VHDL o otros) que puede ser ejecutado en diversas plataformas. Una de las opciones más interesantes es la ejecución del codificador/decodificador en varios procesadores de forma simultánea (sistemas multiprocesador), en los que la carga computacional se puede dividir entre los procesadores del sistema.
RVC utiliza un lenguaje de flujo de datos donde aparecen actores y *tokens*, que tendrán una relevancia más que notable en el desarrollo de este proyecto.

Los actores de un decodificador son el conjunto de operadores que forman el decodificador y mediante el estado de dichos actores se transforman los flujos de entrada de objetos de datos (*tokens*) en flujos de salida.
Los actores realizan su cálculo en una secuencia de pasos que se denominan disparos (*firings*), consumiendo señales (*tokens*) de entrada, modificando el estado y/o distribución de los actores y obteniendo *tokens* de salida.

Por consiguiente, especificar un actor implica describir su interfaz hacia el exterior, los puertos, la estructura de su estado interno, así como los pasos que se pueden llevar a

cabo (en términos de producción y el consumo de contadores, y la actualización del estado actor), y cómo elegir la forma en la que se interpretará dicho actor.

Hasta la fecha este reparto de los actores del decodificador entre los diferentes núcleos del sistema se realiza de forma manual por parte del diseñador, siendo una tarea larga y costosa. El grado de dificultad a la hora de realizar la asignación de los diferentes actores entre los diferentes núcleos, depende del número de actores que posea el decodificador y el número de núcleos sobre los que se esté trabajando; a medida que cualquiera de los dos factores aumenta, mayor es la dificultad a la hora de realizar dicha asignación, llegando a tener miles de posibilidades válidas. Lo que el diseñador pretende es coger cada uno de los actores del decodificador y redistribuirlos entre el los núcleos del sistema de la mejor forma posible. Todo ello implica tener que disponer de un conocimiento profundo de los módulos que componen el sistema desarrollado para lograr una buena distribución, pero sin haber realizado todas las pruebas o combinaciones posibles, con lo que se podrían obviar las distribuciones con las que se obtendrían resultados de ejecución óptimos.

Debido a esto, este proyecto está enfocado al desarrollo de una herramienta que permita realizar la asignación de cada uno de los actores del descodificador a un núcleo de ejecución de forma automática y eficiente, almacenando las mejores distribuciones obtenidas a partir de los datos de rendimiento.

1.2 Objetivos

Los objetivos de este PFC son los siguientes:

El principal objetivo del proyecto es el desarrollo de una aplicación que sea capaz de automatizar el proceso de asignación de los diferentes actores que forman un sistema de descodificación de vídeo basado en el estándar RVC a los núcleos de un sistema multiprocesador para maximizar el rendimiento del sistema desarrollado y almacenar las mejores distribuciones realizadas.

Hay que tener en cuenta que RVC está en continuo desarrollo y por lo tanto se producen cambios con mucha frecuencia. Esto ha implicado estar continuamente pendiente de la evolución producida en los diferentes estándares de codificación sobre los que se ha trabajado, teniendo que realizar progresivas correcciones o cambios a lo largo del desarrollo del programa.

Una vez desarrollada la aplicación se pretende realizar pruebas con diferentes estándares de codificación de vídeo para confirmar su funcionamiento. Una vez hecho esto, se realizará un banco de test con cada uno de los estándares de codificación de vídeo utilizados, sobre varios procesadores (núcleos) y si fuese posible utilizando diversas plataformas hardware para validar su funcionamiento en otros dispositivos.

1.3 Fases del desarrollo

En este apartado, se expondrán las sucesivas fases que se han seguido para el desarrollo de este Proyecto Fin de Carrera.

- Realizar un estudio de RVC para adquirir los conocimientos necesarios para poder abordar este proyecto.
- Analizar los diferentes estándares de codificación de vídeo así como los decodificadores utilizados (MPEG 4 parte 2, MPEG 4 parte 10 CBP y PHP, HEVC).
- Realizar pruebas con los diferentes decodificadores para analizar y comprender cada uno de los actores que componen cada uno de los decodificadores.
- Desarrollar la aplicación.
- Realizar pruebas de funcionamiento.
- Obtener resultados y crear un banco de test con las pruebas realizadas para analizar el comportamiento de la aplicación desarrollada.

CAPÍTULO 2. Estado del arte

Reconfigurable Video Coding (RVC) es un estándar promovido por MPEG, en colaboración con más grupos, para proporcionar un marco innovador de desarrollo de codificación de vídeo. Este marco ofrece una manera de superar la falta de interoperabilidad entre los muchos y diversos códec de video que actualmente hay en el mercado. Cualquier códec de RVC se describe utilizando una programación de flujo de datos que otorga la flexibilidad anteriormente descrita y que aporta la capacidad de reutilización de los módulos ya descritos. Actualmente, dos normas han sido definidas y producidas por el grupo de trabajo RVC:

1- Una representación de la configuración de los codecs (ISO / IEC 23001-4 o MPEG-B pt. 4) [**1**] que describe el formato con el que un decodificador RVC puede ser definido como una red de bloques de cálculo, así como un lenguaje textual propio para describir la definición de los bloques de codificación de vídeo.

2- Una biblioteca de herramientas de vídeo (ISO / IEC 23002-4 o MPEG-C pt. 4) [**2**] que estandariza los actores necesarios para describir los estándares de codificación de video existentes (actualmente MPEG-4 Parte 2 y MPEG-4 parte 10); y los estándares ya desarrollados actualmente como son "SVC y HEVC".

En este capítulo se realizará una breve descripción del estándar RVC, tanto de sus orígenes como de sus motivaciones. También se analizará su escenario de codificación así como sus lenguajes y herramientas de trabajo.

2.1 Origen de RVC

El trabajo y estudio en RVC comenzó en marzo de 2004, durante una reunión del estándar MPEG en Múnich con la investigación de los elementos comunes de todos los estándares MPEG existentes hasta la fecha. Después de dos años de trabajo, se encontró que aunque las especificaciones más distintivas o principales de cada estándar son diferentes, también se encontraron muchas similitudes entre sus arquitecturas y el flujo de datos. Durante la 76ª reunión de MPEG en Montreux (Francia) se realizó una convocatoria de propuestas, en la que trataron el tema de recopilar diversas tecnologías para describir las formas con las que unificar la tecnología de vídeo MPEG.

En la siguiente reunión, ya con las descripciones para realizar la unificación, se propuso crear un marco MPEG Reconfigurable Video Coding. Esta propuesta fue aceptada y se comenzó a trabajar en el desarrollo de los componentes estándares que posteriormente darían lugar al estándar RVC.

Un gran número de normas de codificación de vídeo que se han desarrollado desde el primer estándar MPEG-1 en 1988 han tenido un gran éxito. Los numerosos esfuerzos para conseguir la estandarización, además de tener como primer objetivo garantizar la interoperabilidad entre los diferentes sistemas de compresión de vídeo también pretenden proporcionar las formas adecuadas de las especificaciones para su amplía y fácil expansión en el mercado. Este hecho, presenta un obstáculo debido a que los estándares de vídeo son cada vez más complejos y hace que sea más difícil para los organismos de normalización poder producir las especificaciones necesarias para garantizar la estandarización e interoperabilidad de los nuevos estándares.

El comité MPEG ISO/IEC, Reconfigurable Video Coding (RVC) tiene por objeto tratar lo siguiente:
"Hacer que la producción de nuevos estándares de codificación sea más rápida y eficiente consiguiendo que los dispositivos de vídeo basados en dichas normas, puedan exhibir y ofrecer una mayor flexibilidad con respecto a las tecnologías de codificación empleadas para el contenido del vídeo. Esta idea tiene como finalidad crear o construir una Biblioteca con todos los componentes de codificación de vídeo, en lugar de tener muchos decodificadores de vídeos completos y específicos. De esta manera, el estándar podrá evolucionar con más facilidad y flexibilidad, mediante la extensión que los dispositivos de la biblioteca ofrecen, así como las diferentes configuraciones de vídeo pueden ayudar, mediante variedades de algoritmos de codificación, para crear diferentes codificadores y decodificadores a partir de esta biblioteca predefinida de módulos de codificación". [**4**]

La Fig.1 presenta la evolución temporal de RVC.

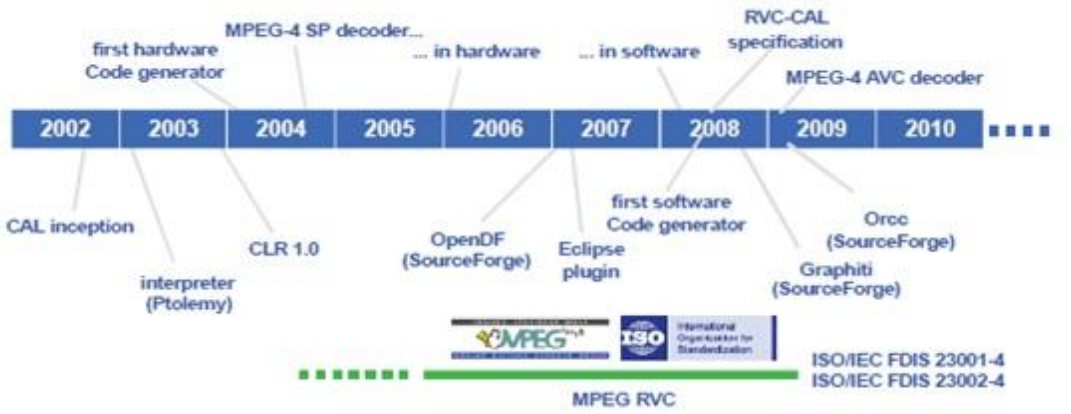


Figura 1. Línea temporal del desarrollo del estándar RVC

2.2 Motivación del estándar RVC

El desarrollo de RVC fue motivado por lo siguiente:

"En las últimas dos décadas, muchos estándares de codificación de vídeo (MPEG-2, MPEG-4, H264, AVC entre otros muchos) se han definido y especificado para satisfacer las necesidades de los consumidores en el mercado. El desarrollo de nuevas tecnologías ha supuesto la utilización de algoritmos cada vez más complejos, pero que generalmente comparten algunas partes comunes (el ejemplo más común es la utilización de la Transformada Directa del Coseno, DCT). Hasta este momento, no había ninguna forma estándar de desarrollo con la que poder explotar y obtener mejor rendimiento de éstas partes comunes. Por este motivo surgió la especificación de un estándar de codificación de vídeo a través de un documento de texto y un software inicial de referencia. Esta primera especificación se pensó para plataformas de un solo núcleo, por lo que posteriormente se modificaría para realizar su expansión a plataformas multinúcleo o plataformas hardware". [**4**]

En el siguiente apartado se ofrecerá una visión más general de los conceptos y las tecnologías de construcción del marco RVC estándar.

2.3 Requisitos y justificación del marco RVC estándar

Como ya se ha comentado en el apartado anterior, MPEG RVC es un nuevo estándar ISO, que actualmente se encuentra en su última etapa de normalización, con el objetivo de proporcionar a través de una biblioteca de componentes las especificaciones de los diferentes códec de vídeo. RVC resuelve esto mediante la definición de dos estándares:

Un lenguaje con el que se puede describir un decodificador de vídeo (RVC-CAL) y una biblioteca de herramientas de codificación de vídeo empleados en MPEG.

La posibilidad de una configuración dinámica, así como la reconfiguración de los códec también requiere nuevas tecnologías y nuevas herramientas para poder describir la nueva sintaxis de flujo de datos y los programas de análisis de estos nuevos códec.

La configuración dinámica se basa en la modificación de grupos de actores (según su funcionalidad) del decodificador así como de su distribución en los diferentes núcleos sobre los que se ejecuta, para así obtener nuevas configuraciones y resultados en tiempo de ejecución.
Los conceptos básicos y esenciales del marco RVC son los siguientes:

2.3.1 Escenario de codificación/decodificación MPEG RVC

Para explicar cuál sería el escenario de codificación/decodificación [**6**] se va a hacer uso de la Fig.2, que define los elementos esenciales del marco RVC.

Está compuesto de tres elementos principales: una normativa Video Tool Library (VTL), una descripción normativa de la conexiones entre las herramientas de codificación (FUs) llamadas Unidades Funcionales de Descripción (FND), que se escribe en la Unidad Funcional de Lenguaje de Red (Network) (RVC-FNL), y una Descripción de la normativa de la estructura de la corriente de bits, es decir, flujo de bits denominada Bitstream Syntax Description (BSD).

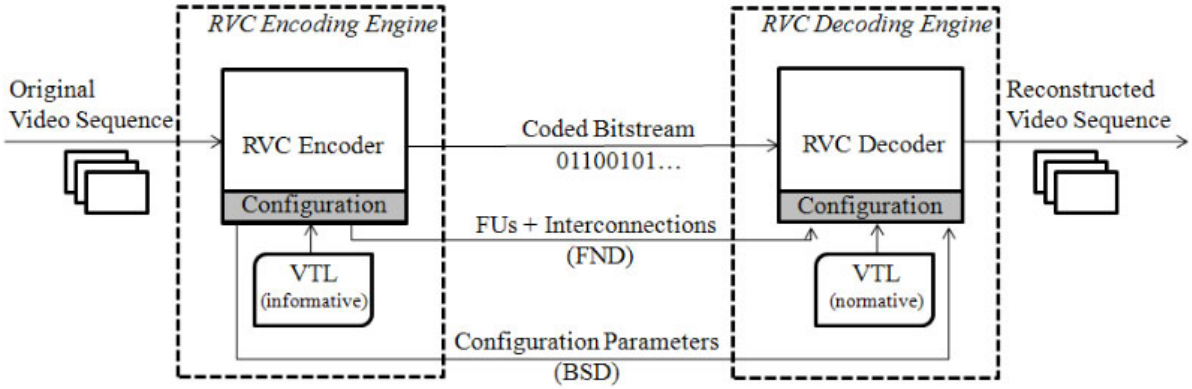


Figura 2. Escenario de codificación del estándar RVC

2.3.2 RVC-CAL. Actor Language

RVC-CAL Actor Language [**5**] es un lenguaje de flujo de datos que describe la funcionalidad de las FUs (Unidades funcionales). Este lenguaje se encarga de describir y definir el comportamiento de los componentes de los flujos de datos llamados actores. Un actor de un decodificador es un componente modular que encapsula su propio estado; esto hace que no se pueda acceder ni modificar el estado de cualquier otro actor. Un actor realiza una acción mediante una secuencia de pasos denominados *firings* (disparos). Durante una ejecución, la evaluación se realiza mediante una acción, elegida entre las varias acciones que un actor puede tener. Una acción define la cantidad de *tokens* (indicadores) consumidos en la entrada, puede cambiar el estado del actor y puede usar *tokens* de salida en función de los *tokens* de entrada y el estado del actor.

La ejecución en el interior de un actor es puramente secuencial: en cualquier punto en el tiempo, una sola acción puede estar activa dentro del actor. Una acción puede consumir o leer *tokens*, modificar el estado interno del actor, generar *tokens* e interactuar con la plataforma en la que se está ejecutando en actor.

La Fig. 3 presenta el modelo general para crear instancias de un decodificador en RVC. En ella, el bloque MPEG-B define los lenguajes que se utilizan para construir el marco MPEG-RVC.

El flujo de datos del lenguaje de programación RVC-CAL es el núcleo del sistema, ya que describe el comportamiento de cada FU. A la interacción de cada una de las FUs con las normas del lenguaje CAL se le une el bloque MPEG-C el cual define la biblioteca de herramientas de codificación de video VTL.

Lo que la Fig.3 ilustra a nivel general es el concepto de que a partir de cualquier modelo de decodificador constituido por FUs y una descripción de la red de decodificación se puede obtener soluciones para la implementación hardware o software.

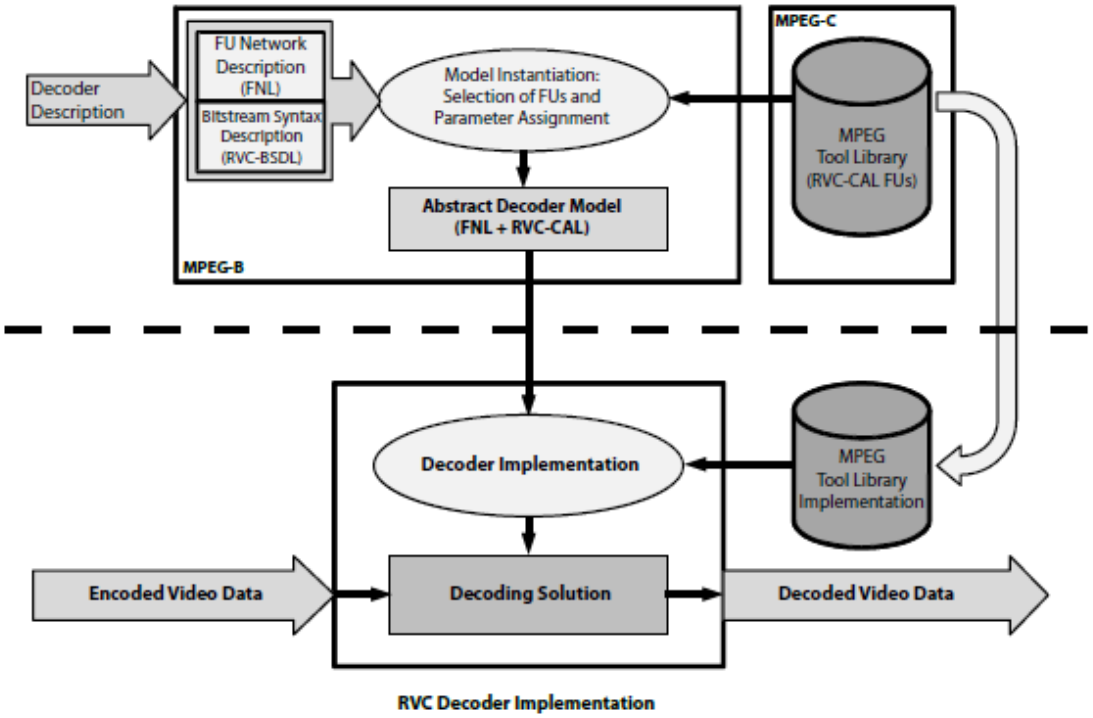


Figura 3. Implementación de un decodificador RVC mediante el estándar RVC

La Fig.4 muestra el modelo computacional de CAL. En ella se muestra el posible flujo de cambio del estado los actores a través de sus acciones y estados.

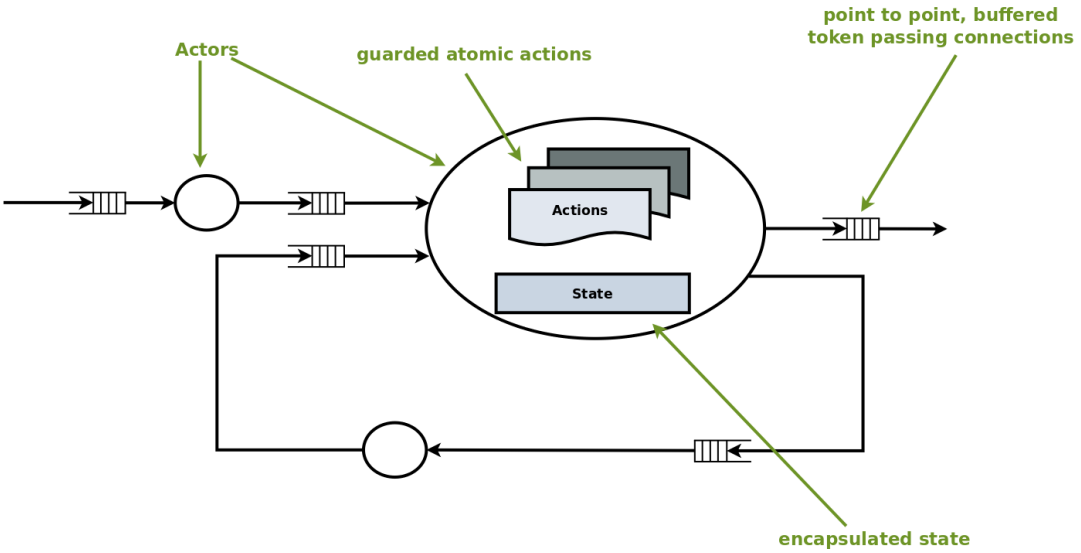


Figura 4. Modelo computacional de CAL

Para analizar la estructura de decodificación de los actores, se va a hacer uso de la Fig.5, que muestra un ejemplo, en el que los actores de decodificación decodifican una imagen y almacenan la imagen decodificada para posteriormente usarla en el proceso de predicción inter.

Cada actor posee la memoria necesaria para almacenar imágenes y encapsularlas en el *Decoded Picture Buffer* (DPB). El actor DBP también contiene el *Deblocking Filter*, siendo una solución de amortiguación para regular y organizar el flujo de vídeo resultante, de acuerdo con la *Memory Management Control Operation* (MMCO) de entrada.

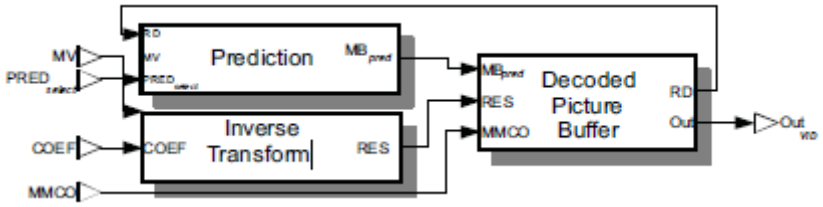


Figura 5. Estructura de los actores de decodificación

El DPB crea cada *frame* sumando los datos de predicción, siempre generados por el actor de predicción, y los datos residuales generados por el actor de Transformada Inversa. El actor de predicción engloba los modos inter e intra así como un multiplexor que envía los datos de la predicción al puerto de salida correspondiente. La entrada $PRED_{select}$ tiene la función de alimentar al conjunto de actores encargados de la predicción.

El objetivo global de esta estructura es cambiar cómodamente entre las configuraciones del decodificador añadiendo o eliminando actores de predicción. En la siguiente Fig.6 se muestra un actor de predicción.

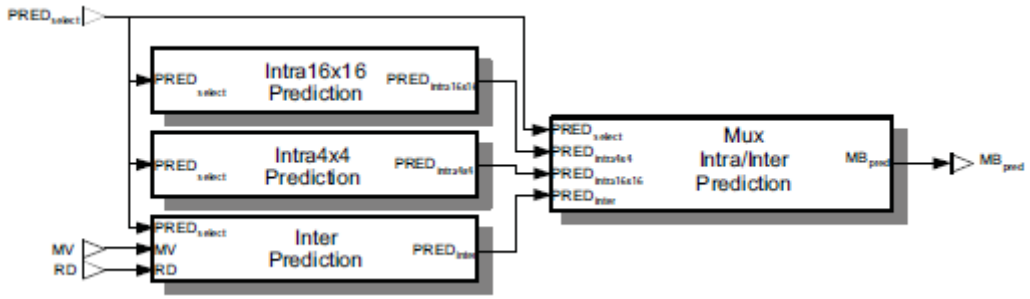


Figura 6. Estructura del actor de predicción

2.3.3 FNL (Unidad funcional de lenguaje de red)

FNL es un lenguaje que describe las diferentes configuraciones de los códec de vídeo. Se trata de un lenguaje XML que enumera las unidades funcionales que componen el códec, los parámetros de los que se componen estas FUs y las conexiones entre dichas FUs. FNL permite construcciones jerárquicas en las que una FU puede ser definida como una composición de otra FU y descrita por otra FND (FU de descripción de red).

2.3.4 BSDL (Bitstream syntax description language)

BSDL es un lenguaje que describe la estructura del flujo de bits de entrada. Se trata de un lenguaje XML que enumera la secuencia de los elementos de sintaxis, que pueden ser modificados en presencia de diversos elementos, de acuerdo con el valor de los elementos previamente decodificados.

La Fig.7 Muestra de forma general la estructura de este lenguaje. Partiendo de un *bitstream* escalable se obtiene un fichero de descripción XML. Partiendo de este fichero y mediante un conjunto de ficheros de estilo XSLT se especifica la presentación de una clase de documentos XML con la descripción de cómo se realiza la transformación de una instancia en un documento XML en diferentes lenguajes de codificación. Con esto se obtiene un *bitstream* escalable adaptado.

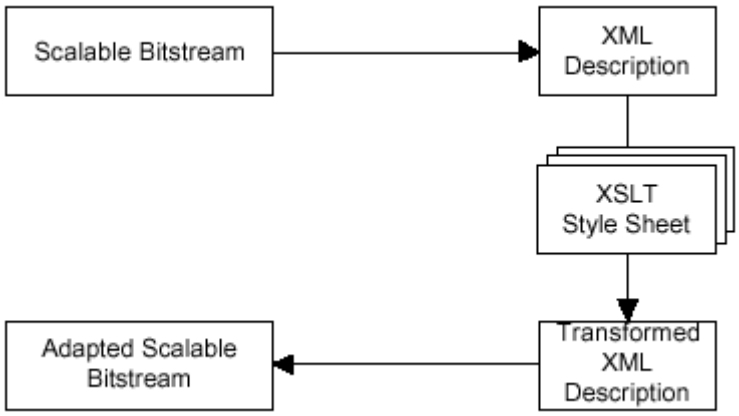


Figura 7. Estructura BSDL

2.3.5 Biblioteca RVC

Se dispone de una biblioteca de herramientas de codificación de vídeo [**16**] compuesta por FUs, que contiene todos los estándares MPEG (MPEG Toolbox). Esta biblioteca se especifica utilizando el lenguaje anteriormente descrito denominado RVC-CAL (subconjunto del lenguaje original CAL) y que principalmente se utiliza como lenguaje de descripción o especificación para cada FU.

En la siguiente Fig.8 se podrá observar la librería de herramientas así como el interior de una FU.

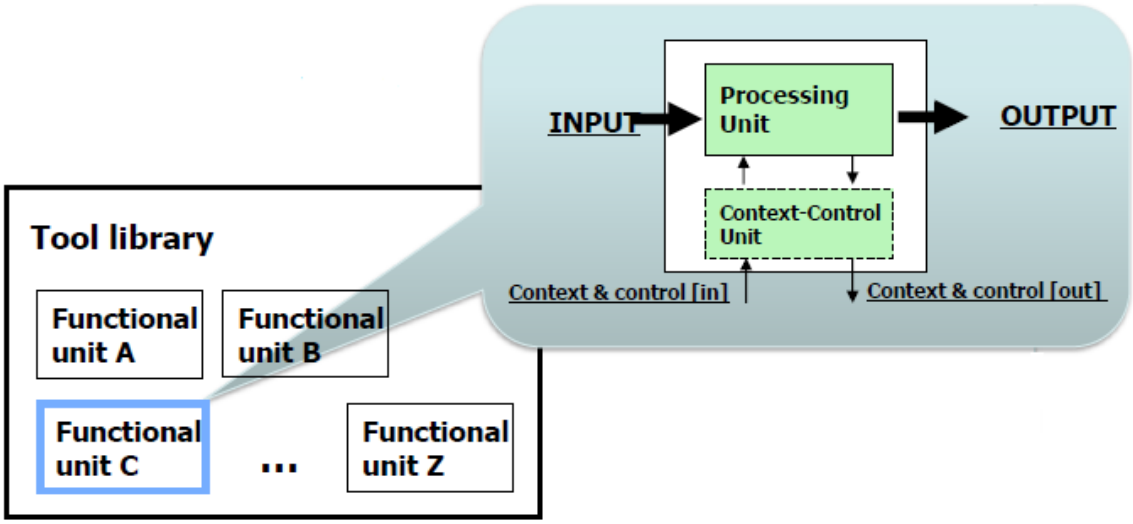


Figura 8. Librería de herramientas

2.3.6 Herramientas RVC

Una de las características destacadas de RVC es que posee herramientas específicas muy útiles a la hora de validar y simular el comportamiento de la *Abstract Decoder Model* (ADM), y herramientas (backends) para la generación automática tanto de software como de hardware de las descripciones de la ADM.

Uno de los objetivos de RVC es realizar decodificadores partiendo de un conjunto de especificaciones de abstracción de alto nivel. Un ADM será un modelo de decodificador de alto nivel descrito como un diagrama de flujo de datos. A partir de él, las herramientas generan código para las diferentes plataformas destino. En nuestro caso se genera código C que puede paralelizarse en varios procesadores.

Las ventajas que ofrece este tipo de modelo de alto nivel son:

- *Abstract Programming Language:* es un lenguaje de programación de alto nivel, que permite a los desarrolladores centrarse en los principales algoritmos de compresión, sin tener que estar pendiente de la programación y arquitectura de bajo nivel
- Paralelismo escalable: se puede obtener el máximo rendimiento de ejecución posible mediante la paralelización de los programas desarrollados debido a los nuevos métodos de codificación y programación de flujo de datos. Se permite que cada actor se pueda ejecutar en un núcleo del procesador.
- Reutilización de código: puesto que la mayoría de las tecnologías de codificación de vídeo poseen muchos elementos comunes, descritos mediante el marco MPEG RVC, se pueden utilizar proyectos y partes de código para desarrollar nuevos trabajos y normas.
- Modularidad de los bloques de codificación: debido al alto nivel de encapsulado al que se someten las herramientas de codificación, la descripción del flujo de datos permite y favorece la configuración del decodificador.

Todo esto, facilita y simplifica la codificación de los flujos de trabajo, para conseguir mejorar y evolucionar a la hora de desarrollar codificadores/decodificadores de vídeos más eficientes.

CAPITULO 3. Formatos de compresión de vídeo

En este capítulo, se va a estudiar y analizar los dos estándares de compresión utilizados en este proyecto, el estándar H.264/MPEG-4 Advanced Video Coding (AVC) y el estándar High Efficiency Video Coding (HEVC). En ambos casos se estudiaran su historia, estandarización, codificación, características así como las ventajas que ofrecen cada uno de ellos de manera resumida.

3.1 H.264/MPEG-4 AVC

Se trata de una norma que define un tipo de códec de video de alta compresión. Este estándar de compresión fue desarrollado por dos grupos, ITU-T Video Coding Experts Group (VCEG) e ISO/IEC Moving Picture Experts Group (MPEG). La motivación del proyecto H.264 fue la de crear un estándar capaz de proporcionar una buena calidad de imagen que tuviera tasas binarias notablemente inferiores a los estándares anteriores (MPEG-2, H.263 o MPEG-4 Part2) y que no incrementase en gran medida la complejidad de su diseño con respecto a los anteriores.

Para que este proyecto se desarrollase lo más rápidamente posible se unieron los grupos ITU-T e ISO/IEC creando el equipo de desarrollo Joing Video Team (JVT), formado por expertos del VCEG y MPEG, en diciembre de 2001, con el objetivo final de completar el estándar en dos años. La ITU-T tenía pensado adoptar el estándar bajo el nombre de ITU-H.264 e ISO/IEC bajo el nombre de MPEG-4 Parte 10 AVC; de ésta unión salió el nombre actualmente conocido como H.264/MPEG-4 AVC.

Para empezar a programar el nuevo código, se adaptaron las siguientes premisas:

- La estructura DCT + Compensación de Movimiento de las versiones anteriores era superior a otros estándares y por esto no había ninguna necesidad de hacer cambios fundamentales en la estructura.
- Algunas formas de codificación de vídeo que habían sido excluidas en el pasado debido a su complejidad y su alto coste de implementación se volverían a examinar para su inclusión puesto que la tecnología VLSI había sufrido un adelanto considerable y una bajada de costes de implementación.
- Para permitir una libertad máxima en la codificación y evitar restricciones que comprometan la eficiencia, no se contempla mantener la compatibilidad con normas anteriores.

3.1.1 Características

El H.264 es un estándar nacido a partir de otros dos anteriores, pero haciendo uso de las redundancias espaciales, temporales y psico-visuales para mejorar la eficiencia de la codificación de vídeo pero con importantes diferencias y mejoras respecto a ellos.

Las nuevas características más significativas con las que consta este estándar respecto a sus antecesores son las siguientes:

- Codificación de entropía mejorada.
- Compensación/predicción de movimiento mejorada.
- Pequeños bloques para la codificación por transformada.
- Filtro "*Deblocking*" mejorado.
- Mejora significativa del ahorro del *bitrate* (aproximadamente un 50%), manteniendo la misma calidad de imagen y la relación S/N.

Debido a la mejora de estas características se han producido codificadores/decodificadores más desarrollados en cuanto a tamaño y complejidad, obteniendo así más eficiencia en la decodificación de vídeos.

3.1.2 Niveles

En la Fig.9 se pueden observar los diferentes niveles de este estándar así como las características que poseen cada uno de ellos. En éste caso se ha utilizado como ejemplo el estándar H.264/MPEG parte 10 AVC.

Número de nivel	Max. macrobloques por segundo	Max. tamaño de trama (macrobloques)	Max. video bit rate (VCL) para Baseline, Extended and Main Profiles	Max. video bit rate (VCL) para High Profile	Max. video bit rate (VCL) para High 10 Profile	Max. video bit rate (VCL) para High 4:2:2 and High 4:4:4 Predictive Profiles	Ejemplos para alta resolución @ frame rate (max stored frames) in Level
1	1485	99	64 kbit/s	80 kbit/s	192 kbit/s	256 kbit/s	128x96@30.9 (8) 176x144@15.0 (4)
1b	1485	99	128 kbit/s	160 kbit/s	384 kbit/s	512 kbit/s	128x96@30.9 (8) 176x144@15.0 (4)
1.1	3000	396	192 kbit/s	240 kbit/s	576 kbit/s	768 kbit/s	176x144@30.3 (9) 320x240@10.0 (3) 352x288@7.5 (2)
1.2	6000	396	384 kbit/s	480 kbit/s	1152 kbit/s	1536 kbit/s	320x240@20.0 (7) 352x288@15.2 (6)
1.3	11880	396	768 kbit/s	960 kbit/s	2304 kbit/s	3072 kbit/s	320x240@36.0 (7) 352x288@30.0 (6)
2	11880	396	2 Mbit/s	2.5 Mbit/s	6 Mbit/s	8 Mbit/s	320x240@36.0 (7) 352x288@30.0 (6)
2.1	19800	792	4 Mbit/s	5 Mbit/s	12 Mbit/s	16 Mbit/s	352x480@30.0 (7) 352x576@25.0 (6)
2.2	20250	1620	4 Mbit/s	5 Mbit/s	12 Mbit/s	16 Mbit/s	352x480@30.7(10) 352x576@25.6 (7) 720x480@15.0 (6) 720x576@12.5 (5)
3	40500	1620	10 Mbit/s	12.5 Mbit/s	30 Mbit/s	40 Mbit/s	352x480@61.4 (12) 352x576@51.1 (10) 720x480@30.0 (6) 720x576@25.0 (5)
3.1	108000	3600	14 Mbit/s	17.5 Mbit/s	42 Mbit/s	56 Mbit/s	720x480@80.0 (13) 720x576@66.7 (11) 1280x720@30.0 (5)
3.2	216000	5120	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@60.0 (5) 1280x1024@42.2 (4)
4	245760	8192	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@68.3 (9) 1920x1088@30.1 (4) 2048x1024@30.0 (4)
4.1	245760	8192	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1280x720@68.3 (9) 1920x1088@30.1 (4) 2048x1024@30.0 (4)
4.2	522240	8704	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1920x1088@64.0 (4) 2048x1088@60.0 (4)
5	589824	22080	135 Mbit/s	168.75 Mbit/s	405 Mbit/s	540 Mbit/s	1920x1088@72.3 (13) 2048x1024@72.0 (13) 2048x1088@67.8 (12) 2560x1920@30.7 (5) 3680x1536/26.7 (5)
5.1	983040	36864	240 Mbit/s	300 Mbit/s	720 Mbit/s	960 Mbit/s	1920x1088@120.5 (16) 4096x2048@30.0 (5) 4096x2304@26.7 (5)
3	40500	1620	10 Mbit/s	12.5 Mbit/s	30 Mbit/s	40 Mbit/s	352x480@61.4 (12) 352x576@51.1 (10) 720x480@30.0 (6) 720x576@25.0 (5)
3.1	108000	3600	14 Mbit/s	17.5 Mbit/s	42 Mbit/s	56 Mbit/s	720x480@80.0 (13) 720x576@66.7 (11) 1280x720@30.0 (5)
3.2	216000	5120	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@60.0 (5) 1280x1024@42.2 (4)
4	245760	8192	20 Mbit/s	25 Mbit/s	60 Mbit/s	80 Mbit/s	1280x720@68.3 (9) 1920x1088@30.1 (4) 2048x1024@30.0 (4)
4.1	245760	8192	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1280x720@68.3 (9) 1920x1088@30.1 (4) 2048x1024@30.0 (4)
4.2	522240	8704	50 Mbit/s	62.5 Mbit/s	150 Mbit/s	200 Mbit/s	1920x1088@64.0 (4) 2048x1088@60.0 (4)
5	589824	22080	135 Mbit/s	168.75 Mbit/s	405 Mbit/s	540 Mbit/s	1920x1088@72.3 (13) 2048x1024@72.0 (13) 2048x1088@67.8 (12) 2560x1920@30.7 (5) 3680x1536/26.7 (5)
5.1	983040	36864	240 Mbit/s	300 Mbit/s	720 Mbit/s	960 Mbit/s	1920x1088@120.5 (16) 4096x2048@30.0 (5) 4096x2304@26.7 (5)
Número de niveles	Max. macrobloques por segundo	Max. tamaño de trama (macrobloques)	Max. video bit rate (VCL) for Baseline, Extended and Main Profiles	Max. video bit rate (VCL) for High Profile	Max. video bit rate (VCL) for High 10 Profile	Max. video bit rate (VCL) for High 4:2:2 and High 4:4:4 Predictive Profiles	Ejemplos para alta resolución @ frame rate (max stored frames) in Level

Figura 9. Niveles H.264/MPEG parte 10 AVC

3.1.3 Tipos de imágenes

En esta norma se pueden encontrar las mismas imágenes que en las normas anteriores (Imágenes I, P y B) y dos nuevas, la SP (Switching P) y la SI (Switching I) que sirven para codificar la transición entre dos flujos de vídeo. Esto permite pasar de un vídeo a otro utilizando predicción temporal o espacial con la ventaja de poder realizar la reconstrucción de valores específicos de la muestra independientemente de que se utilicen imágenes de referencia diferentes o un número diferente de imágenes de referencia en el proceso de predicción. Este hecho se consigue sin la necesidad de enviar imágenes intra con un gran peso en tiempos de procesamiento.

En la Fig.10 se puede observar un resumen de los tipos de imágenes comentados en este apartado con una breve descripción y el perfil que poseen cada una de ellas.

Tipo de Slice	Descripción	Perfil
***I** (Intra)*	Contiene sólo macrobloques I (predichos de datos codificados previamente dentro del mismo slice, lista 0 ó pasados cercanos)	Todos
***P** (Predictivo)*	Contiene macrobloques P (predichos desde una imagen de referencia) y/o macrobloques I	Todos
***B** (Bi-Predictivo)*	Contiene macrobloques B y/o macrobloques I (predichos desde listas 0 – pasados cercanos y 1 – futuros cercanos)	Extendido y Principal
SP (Switching P) *Novedad frente a MPEG-2*	Facilita transición entreflujos o *streams* codificados, contiene macrobloques P y/o I	Extendido
SI (Switching I) *Novedad frente a MPEG-2*	Facilita transición entre flujos o *streams* codificados, contiene macrobloques SI (un tipo especial intra frame)	Extendido

Figura 10. Resumen de tipos de imágenes

3.1.4 Compensación de movimiento

El proceso de compensación de movimiento propone un cambio notable con respecto a las normas anteriores debido a que propone una gran variedad de formas y de particiones de bloques. Cada macrobloque, independientemente del tamaño original (16x16 píxeles), puede ser descompuesto en sub-bloques más pequeños de 16x8, 8x16 u 8x8 píxeles, éste último pudiendo ser descompuesto a su vez en 8x4, 4x8 ó 4x4. Esta variedad de sub-bloques y particiones proporcionan una mayor exactitud en la estimación, a lo que se suma una precisión que puede llegar hasta ¼ de píxel.

3.1.5 Transformada

La transformada que se utiliza en este estándar es una aproximación a la DCT (Transformada Directa del Coseno) con las siguientes particulares:

- Tamaño**:** 4x4 píxeles (8x8 en los perfiles FRExt).
- Coeficientes enteros: lo que permite evitar los errores de redondeo habituales en la DCT clásica (coeficientes irracionales) y garantizar un ajuste perfecto entre la transformación directa y la inversa.
- Precisión finita: otra consecuencia favorable de la característica anterior es que se puede calcular sin exceder los 16 bits de precisión.
- Eficiencia: Se puede implementar exclusivamente por medio de sumas y desplazamientos binarios.

La Fig.11 muestra un ejemplo de una matriz de transformación a través de la cual podemos obtener las ecuaciones de luminancia. Mediante otras transformadas se pueden obtener también la crominancia y los datos residuales.

$$\mathbf{Y} = \mathbf{AXA}^T = \begin{bmatrix} a & a & a & a \\ b & c & -c & -b \\ a & -a & -a & a \\ c & -b & b & -c \end{bmatrix} \mathbf{X} \begin{bmatrix} a & b & a & c \\ a & c & -a & -b \\ a & -c & -a & b \\ a & -b & a & -c \end{bmatrix}$$

Figura 11. Ejemplo de matrices de transformación para obtener la luminancia

Continuando con el ejemplo anterior, las ecuaciones de luminancia que se obtendría serían las siguientes:

$$a = \frac{1}{2}$$

$$b = \sqrt{\frac{1}{2}} \cos\left(\frac{\pi}{8}\right)$$

$$c = \sqrt{\frac{1}{2}} \cos\left(\frac{3\pi}{8}\right)$$

3.1.6 Cuantificación

Cada paso del parámetro de cuantificación (QP) incrementa un 12,5% el intervalo de cuantificación, lo que equivale a duplicarlo por cada 6 pasos. El rango dinámico del QP ha aumentado respecto a normas precedentes, puesto que los valores van de 0 a 51. Los macrobloques se cuantifican utilizando un parámetro de control que puede cambiar adaptándose al bloque menor cada bit adicional (partiendo de 8 bits, 52 pasos). Además, para poder conseguir los mejores resultados visuales la cuantificación de la crominancia se realiza de una forma más eficiente que la de luminancia.

3.1.7 "Deblocking Filter"

Se trata de un filtro anti-bloques que mejora la eficiencia de compresión y la calidad visual de las secuencias de vídeo eliminando los posibles defectos en la codificación.

Este filtro se aplica tanto en el codificador (antes de almacenar macrobloques para futuras predicciones), como en el decodificador (antes de reconstruir y mostrar macrobloques); de este modo se mejora notablemente la compresión.

Este filtro suaviza los bordes de los bloques, mejorando la apariencia de los *frames* y por lo tanto, la de la imagen.
A continuación se puede observar un ejemplo de este filtro mediante la Fig.12.

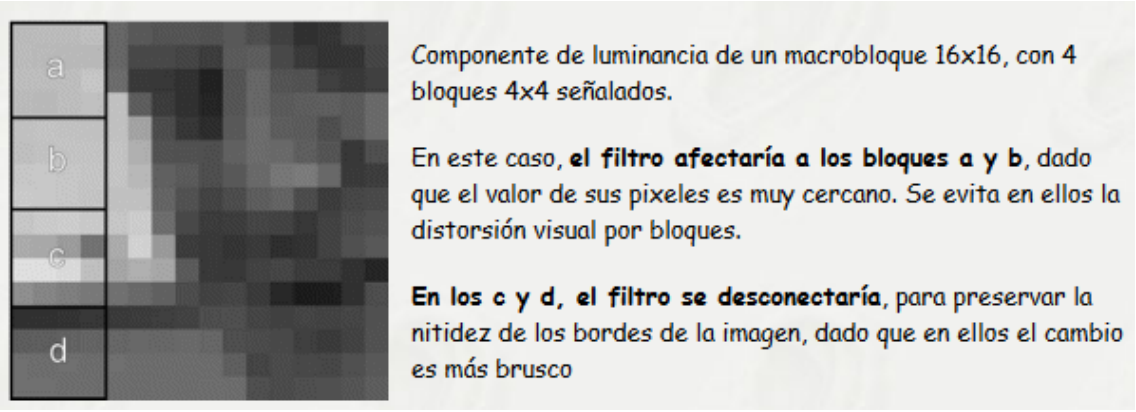


Figura 12. Ejemplo filtro de *Deblocking*

3.1.8 Exploración de los coeficientes

En esta norma existen dos formas diferentes de explorar los coeficientes transformados denominados Zig-Zag y Zig-Zag inverso. El primer tipo de exploración hace referencia a la exploración Huffman [REFERENCIA], mientras que el segundo a la aritmética. Este segundo modo permite la lectura del macrobloque en sentido contrario.
En la Fig.13 se muestran ambos tipos de exploración.

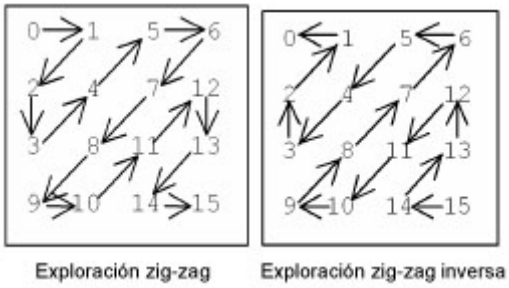


Figura 13. Exploración Zig-Zag y Zig-Zag inversa

3.1.9 Codificación de entropía

La codificación de entropía se puede realizar de tres formas diferentes:

- UVLC (Universal Variable Length Coding): utilizado para codificar la gran mayoría de los elementos de sincronización y cabeceras. Un ejemplo de este tipo de codificación está representado en la Fig.14.

Símbolo	Código
0	1
1	010
2	011
3	00100
4	00101
5	00110
6	00111
7	0001000
8	0001001
...	...

Figura 14. Universal Variable Length Code

- Los otros dos métodos, CABAC (Context Adaptative Binary Arithmetic Codign) y CAVLC (Context Adaptative Variable Length Coding) son utilizados para codificar el resto de elementos sintácticos como los coeficientes o los vectores de movimiento. Este conjunto de métodos se denomina VLC (Variable Length Coding).

En la Fig.15 Se puede observar la codificación por entropía como la etapa final de la compresión.

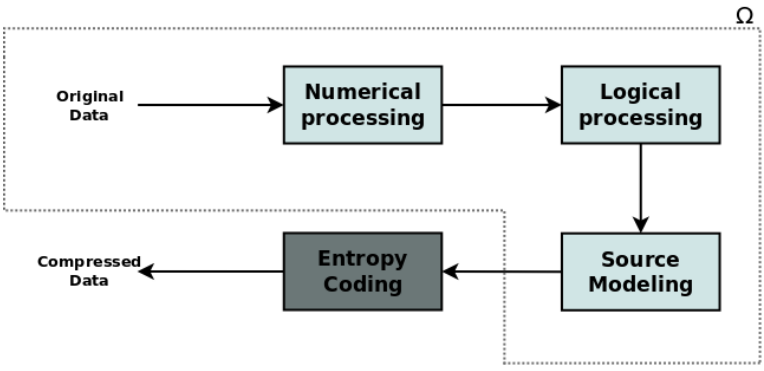


Figura 15. Codificación de entropía

3.1.10 Adaptación a la red

Conceptualmente los algoritmos de codificación están divididos en dos capas: una primera capa de codificación de vídeo VCL (Video Coding Layer) que se ocupa de representar eficazmente el contenido de vídeo y una capa de adaptación a la red NAL (Network Adaptación Layer) que está dirigida más particularmente a adaptar el formato de datos de vídeo al soporte de transmisión.

3.1.11 Algoritmos para la prevención de pérdidas

Los algoritmos para la prevención de pérdidas son….

- FMO y ASO: la ordenación flexible de macrobloques (FMO) y la ordenación arbitraria de *slices* (ASO) son técnicas para reestructurar la representación de las regiones fundamentales (macrobloques) aunque también pueden ser utilizadas para otros objetivos.

- DP: la partición de datos (DP) proporciona la capacidad de separar los elementos de sintaxis más importantes de los menos importantes en paquetes de datos diferentes, permitiendo el uso de protección de error desigual (UEP).

- RSD: el algoritmo de *slices* redundantes (RS) permite a un codificador enviar una representación suplementaria de una región de imagen que puede ser usada si la representación primaria es corrompida o perdida.

3.1.12 Conclusiones

H.264/MPEG-4 AVC supone una gran tecnología con respecto a las normas de codificación de vídeo anteriores. Las diferencias se pueden encontrar a pequeña escala sobre el principio general de codificación (predicción, transformada, cuantificación, etc.). La clave de todo ello es la menor cuantía de información que se necesita almacenar en los videos codificados mediante este códec y la maya de carga computacional utilizada.

En la Fig.16 se ilustra un esquema con un resumen general del códec H.264.

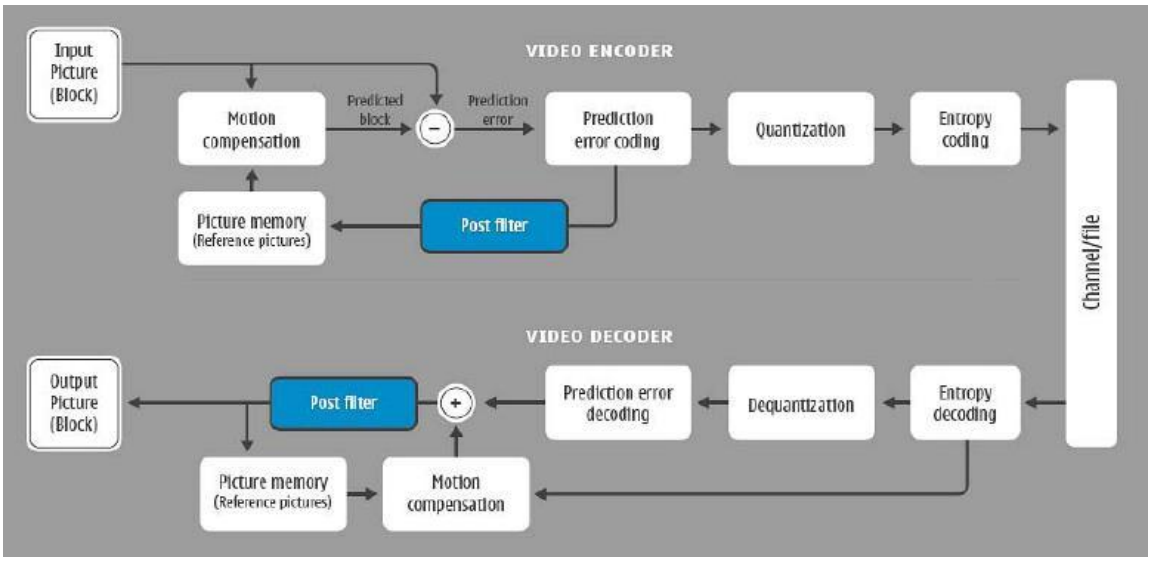


Figura 16. Esquema general H.264

3.2 HEVC: High Efficiency Video Coding

HEVC es un estándar de compresión de vídeo sucesor de H.264/MPEG-4 AVC. En comparación con el formato anterior, HEVC posee el doble de capacidad de compresión de datos pero manteniendo el mismo nivel de calidad de vídeo. Puede ser usado alternativamente para proporcionar una notable mejora en la calidad de vídeo manteniendo la misma velocidad de bits. Esta nueva norma puede ayudar al desarrollo de *Ultra High Definition* (UHD) y a resoluciones de calidad superior como por ejemplo 8K hasta 8192x4320.

La primera versión de éste estándar fue completada y publicada a principios de 2013. Varias extensiones de la tecnología siguen en continuo desarrollo, incluyendo diversas extensiones: de rango, de codificación escalable y de vídeo en 3D.

3.2.1 Historia

En 2004, el ITU-T Grupo de expertos en codificación de vídeo (VCEG) comenzó un importante estudio de los avances tecnológicos que podrían dar lugar a la creación de una nueva norma o estándar de compresión de vídeo. En Octubre de este mismo año, se barajaron y estudiaron varias posibilidades y técnicas para mejorar la norma H.264/MPEG-4 AVC. En la siguiente reunión de VCEG, que se llevó a cabo en Enero de 2005, se comenzó a designar temas específicos como “áreas técnicas clave” (KTA) para una mayor y profunda investigación. Una base de código de software denominado el código base KTA fue establecido para evaluar las propuestas. Dicho software se basó en el modelo de software de referencia mixto (JM), el cual fue desarrollado por MPEG y VCEG Joint Video Team. Tecnologías propuestas fueron integradas en el software KTA y probadas en las evaluaciones experimentales de los próximos cuatro años.

A partir de éste punto se tuvieron en cuenta dos enfoques a la hora de estandarizar la tecnología de compresión mejorada: crear una nueva norma o crear diversas extensiones a la norma en ese momento vigente H.264. El proyecto, en sus primeros pasos contó con dos propuestas de nombres significativas: H.265 o H.NGVC (Codificación de Vídeo de Próxima Generación); y fue una parte importante de la obra de VCEG hasta su evaluación en el proyecto conjunto HEVC con MPEG en el año 2010.

Los requisitos previos necesarios para NGVC fue la capacidad de tener una reducción de la velocidad de bits del 50% manteniendo la misma calidad de imagen subjetiva en comparación con el H.264 de alto perfil y complejidad computacional que van desde ½ a 3 veces la del perfil elevado. La propuesta de NGVC sería la de proporcionar una reducción de la tasa del bits del 25% y una del 50% en complejidad con la misma calidad de vídeo percibida como alto perfil, o para proporcionar una mayor reducción de la tasa de bits con un pequeño incremento en complejidad.

El grupo MPEG (Moving Picture Experts Group ISO/IEC) inició un proyecto similar en 2007, inicialmente llamado Codificación de vídeo de alto rendimiento. El objetivo del proyecto en julio de 2007 era el de obtener una reducción de la tasa de bits del 50% y cuyas primeras evaluaciones se realizaron con las modificaciones del software codificador de referencia KTA desarrollado por VCEG. En julio de 2009, dichos resultados mostraron una reducción de bits de promedio del 20% en comparación con el AVC High Profile. Estos resultados llevaron a MPEG en colaboración con VCEG a fijar sus esfuerzos en la creación de una nueva norma.

Un Conjunto Formal de Propuestas (PPC) en la tecnología de compresión de vídeo se emitió en enero de 2010 por VCEG y MPEG, y las propuestas fueron evaluadas en la primera reunión, que tuvo lugar en abril de ese mismo año, del nuevo grupo de trabajo MPEG y VCEG Equipo Conjunto de Colaboración en materia de Video Coding (JCT-VC). Fueron presentadas un total de 27 propuestas. Las sucesivas evaluaciones mostraron que alguna de estas propuestas podrían alcanzar la misma calidad visual AVC en la mitad de la velocidad de bits en muchas de las pruebas realizadas, a costa de 2 x -10 x aumento de la complejidad computacional; y algunas propuestas lograron una buena calidad subjetiva con resultados de tasa de bits con una menor complejidad computacional que los codificadores de alto perfil de referencia AVC. En dicha reunión, el nombre de High Efficiency Video Coding (HEVC) fue aprobado como nombre final para el proyecto conjunto. A partir de esta reunión, las características integradas JCT-VC de algunas de las mejores propuestas fueron integradas en una sola base de código de software y un "modelo de examen", y realizando a su vez, más experimentos para evaluar varias características propuestas. El primer borrador de la especificación de trabajo de HEVC se produjo en la tercera reunión JCT-VC en octubre de 2010. En posteriores reuniones se realizaron muchos cambios en las herramientas y configuración de la codificación de HEVC.

3.2.2 Eficiencia de codificación

El diseño de la mayoría de las normas de codificación de vídeo está dirigido principalmente a conseguir la mayor eficiencia de codificación.

La eficacia de la codificación es la capacidad de codificar vídeo a la velocidad binaria más baja posible mientras se mantiene un cierto nivel de calidad de vídeo. Hay dos maneras de medir la eficacia de la codificación de vídeo que son: usar una métrica objetiva como la relación señal ruido de pico (PSNR) o utilizar la calidad subjetiva del vídeo. La evaluación subjetiva de la calidad de vídeo se considera que es la manera más importante de medir un estándar de codificación de vídeo.

La eficiencia de codificación del perfil principal de HEVC *Main Profile* (MP) ha sido comparada con muchos perfiles de las normas anteriores, como por ejemplo: H.264/MPEG-4 AVC *High Profile* (HP), MPEG-4 *Advanced Simple Profile* (ASP),

H.263 *High Profile Latency* (HLP) y H.262/MPEG-2 Main Profile (MP). La codificación de vídeo se realizó principalmente para aplicaciones de entretenimiento. Doce *bitrates* se crearon para las nueve secuencias de prueba de vídeo con un codificador de HM-8.0 HEVC. De estas nueve pruebas, cinco de ellas contaban con una resolución de alta definición, mientras que las cuatro restantes se encontraban en formato WVGA (800x480) de resolución.

A raíz de los resultados de las pruebas, la capacidad de compresión así como las reducciones de la tasa de bits para HEVC se determinaron utilizando como referencia de comparación el PSNR.

En la siguiente Fig.17 se puede encontrar una comparación entre los estándares de codificación de vídeo, anteriormente mencionados, empleando el PSNR.

Video coding standard	Average bit rate reduction compared to			
	H.264/MPEG-4 AVC HP	MPEG-4 ASP	H.263 HLP	H.262/MPEG-2 MP
HEVC MP	35.4%	63.7%	65.1%	70.8%
H.264/MPEG-4 AVC HP	–	44.5%	46.6%	55.4%
MPEG-4 ASP	–	–	3.9%	19.7%
H.263 HLP	–	–	–	16.2%

Figura 17. Comparación de los estándares de codificación de video basado en la igualdad de PSNR

HEVC MP también ha sido comparado con el perfil H.264/MPEG-4 AVC HP para una calidad de vídeo subjetiva. Esta codificación de vídeo se hizo para aplicaciones de entretenimiento y cuatro tipos de *bitrates* diferentes se obtuvieron mediante las nueve secuencias de prueba de vídeo [REFERENCIA] utilizando un codificador HEVC HM-5.0. La evaluación subjetiva de las pruebas se realizó anteriormente a las pruebas sobre PSNR y por lo tanto utilizó una versión anterior de codificador HEVC, por lo que tenía un rendimiento ligeramente más bajo. Las reducciones de la tasa de bits se determinan teniendo en cuenta los valores de las puntuaciones medias de opinión, en base a la evaluación subjetiva. La reducción general de la tasa de bits subjetiva HEVC MP en comparación con H.264/MPEG-4 AVC HP fue del 49,3%.

La Escuela Politécnica General de Lausanne (EPFL) realizó un estudio para evaluar la calidad de vídeo subjetiva de HEVC a resoluciones superiores a HDTV. Dicho estudio se realizó con tres resoluciones de vídeo diferentes: 3840x1744 a 24 *frames por segundo (FPS)*, 3840x2048 a 30 FPS y 3840x2160 a 30 FPS. Las siguientes cinco secuencias de vídeo, mostraban gente en la calle, tráfico y una escena de la computadora de código abierto de la película de animación *Sintel*. Estas secuencias fueron elegidas principalmente por la cantidad de movimiento y colorido que poseían así como la velocidad de las animaciones. Estas secuencias se codificaron utilizando el codificador

HM-6.1.1 HEVC y el JM-18.3 H.264/MPEG-4 AVC y con cinco diferentes *bitrates*. Las reducciones de la tasa de bits se determinaron de la misma forma que HEVC MP. El estudió comparó HEVC MP con H.264 MPEG-4 AVC HP y mostró que para HEVC MP basado en la reducción media de bit-rate en PSNR fue del 44,4%, mientras que la reducción media de la tasa de bits basada en la calidad de vídeo subjetiva fue del 66,5%.

En una comparación de rendimiento HEVC lanzado en abril de 2013, el HEVC MP y Main 10 Profile (M10P) se compararon con H.264/MPEG-4 AVC HP y H10P con secuencias de vídeo con resoluciones de 3840x2160. Estas secuencias de vídeo se codifican utilizando el codificador HM-10.0 y el JM-18.4 H.264/MPEG AVC. La reducción del promedio de la tasa de bits basado en PSNR fue del 45% para un vídeo con fotogramas de predicción inter.

3.2.3 Características

HEVC fue diseñado principalmente para mejorar la eficacia de codificación en comparación con H.264 MPEG AVC HP, es decir, para reducir a la mitad los requisitos de velocidad de bits, manteniendo una calidad de imagen aceptable, con el inconveniente de una posible mayor complejidad computacional. HEVC fue diseñado con el objetivo de permitir que el contenido del vídeo pueda tener una relación de compresión de datos de hasta 1000:1. En función de los requisitos necesarios de la aplicación, los codificadores HEVC pueden negociar la tasa de compresión, la resistencia a los errores y el tiempo de retardo de la aplicación, sin tener en cuenta la complejidad computacional. Dos de las características principales o claves, que HEVC mejoró en comparación con H.264 MPEG-4 AVC, fue el apoyo de una mayor resolución de vídeo y los métodos de procesamiento paralelo mejoradas.

HEVC está dirigido a pantallas HDTV de última generación y sistemas de captura de contenidos que ofrecen velocidades de fotogramas escaneado progresivas y resoluciones de pantalla de QVGA (320x240) a 4320p (8192x4320), así como la calidad de imagen mejorada en términos de nivel de ruido, espacios de color y el rango dinámico.

3.2.4 Capa de codificación de vídeo

La capa de codificación de vídeo de HEVC utiliza un enfoque híbrido, en el que se utiliza la predicción de imágenes inter-intra y una codificación 2D mediante transformada. El codificador HEVC divide la primera imagen en regiones en forma de bloques, o bien, crea un punto de acceso aleatorio mediante predicción intra/cuadro. Este tipo de predicción se da en el caso en que la predicción de los bloques en la imagen se basa simplemente en la información en esa foto. Para el resto de imágenes la predicción se realiza tomando la información de predicción de otras imágenes. Una vez finalizado el proceso de predicción, la imagen pasa por un bucle que filtra la representación final de la foto y se almacena en la memoria intermedia de la imagen

descodificada. Esta imagen puede ser posteriormente utilizada para la predicción de otras imágenes.

HEVC fue diseñado con la idea de utilizar el vídeo de exploración progresiva y se tuvieron que añadir nuevas herramientas de codificación específicamente para vídeo entrelazado. Las herramientas de codificación específicas para entrelazado, como MBAFF y PAFF, no se admiten en HEVC. HEVC envía datos de meta-stream que especifica cómo fue enviado el vídeo entrelazado. El vídeo entrelazado puede ser enviado de dos formas diferentes: mediante la codificación de cada campo como una imagen separada o mediante la codificación de cada fotograma como una imagen separada. Esto permite que el vídeo entrelazado a enviar con HEVC no requiera de procesos de decodificación entrelazados especiales para ser añadido a los decodificadores HEVC.

3.2.5 Herramientas de codificación

En este sub-apartado se describen brevemente las herramientas de codificación más importantes de HEVC así como los detalles y características principales de las mismas, pero sin profundizar mucho en ello.

3.2.5.1 Unidad de Árbol de Codificación (CTU)

HEVC reemplaza los macrobloques, utilizados en las normas anteriores, por unidades de árbol de codificación (CTUs). Estas unidades pueden utilizar un bloque de grandes estructuras de hasta 64x64 píxeles y puede sub-dividir, con más eficacia, una imagen en estructuras de tamaño variable. HEVC divide inicialmente la imagen en las unidades de transporte, que pueden ser de 64x64, 32x32 o de 16x16 con un tamaño de bloque más grande, lo que conlleva generalmente un aumento en la eficiencia de la codificación.

3.2.5.2 Procesamiento paralelo

En este sub-apartado se hablará de las diferentes herramientas para realizar el procesamiento paralelo.

- Los *tiles* (bloques) permiten que la imagen sea dividida en una cuadrícula de regiones rectangulares que pueden ser codificadas/decodificadas de forma independiente. El propósito principal de estos *tiles* es permitir el procesamiento paralelo. Las cuadrículas pueden ser decodificadas de forma independiente y pueden permitir el acceso aleatorio a regiones específicas de una imagen en una secuencia de vídeo.
- El procesamiento paralelo de frente de onda (WPP) es cuando un sector está dividido en filas de las unidades de transporte, en el que la primera fila se decodifica normalmente, pero cada fila adicional requiere que las decisiones se

realicen en la fila anterior. WPP contiene la información de uso del codificador de entropía de la que precede a la fila de las unidades de transporte y permite para un método de procesamiento paralelo obtener una mejor compresión de las cuadrículas.

- En HEVC los *tiles* y WPP están permitidos, pero son opcionales, no necesarios. Si los *tiles* están presentes, deben tener al menos 64 píxeles de alto y 256 píxeles de ancho, con un límite específico de nivel en el número de cuadrículas permitidas.
- Las capas pueden ser decodificadas independientemente unas de otras, con la finalidad de la re-sincronización en caso de pérdidas de datos en el flujo de E/S del vídeo. Las divisiones son las unidades de transporte decodificados en el orden de la exploración de trama; se pueden usar diferentes tipos de codificación para cada sector de la imagen como los tipo P o B.
- Las capas de codificación de la imagen que son dependientes unas de otras, pueden permitir que los datos relacionados con tiles o WPP puedan acceder más rápidamente por el sistema en lugar de tener que decodificar una capa completa. El propósito principal de las capas dependientes es la de permitir la codificación de vídeo con bajo retardo, debido a su menor latencia.

3.2.5.3 Codificación de la entropía

HEVC utiliza un contexto de la adaptación de codificación aritmética binaria (CABAC), un algoritmo muy similar al que había en H.264/MPEG 4 AVC. CABAC es el único método de codificación de entropía que se permite en HEVC, mientras que hay dos métodos de codificación de entropía en la norma anterior. Este tipo de algoritmo fue diseñado principalmente para obtener un mayor rendimiento. A continuación se exponen algunas de las mejoras o ventajas de CABAC para HEVC:

- El número de bloques codificados se ha reducido a razón de 8x.
- El modo de derivación CABAC se ha mejorado en términos de su diseño para aumentar el rendimiento.
- Las dependencias entre los datos codificados ha sufrido una variación para conseguir aumentar aún más el rendimiento.
- Se puede seleccionar más de un contexto en el que se consigue un aumento de la eficiencia.

3.2.5.4 Predicción intra

H.264/MPEG 4 AVC especifica 8 modos diferentes de dirección para la predicción intra específica; HEVC ha dado un salto en referencia a esto consiguiendo 33 modos de dirección. Los modos de predicción intra utilizan datos a partir de bloques adyacentes de predicción que han sido decodificadas previamente.

3.2.5.5 Compensación de movimiento

Para la interpolación de posiciones de muestras fraccionales luma utiliza una interpolación unidimensional media, con un filtro de interpolación de 8 *taps* (etapas) o 1/4 de muestra con 7 *taps*. HEVC ha mejorado la precisión de filtro debido al aumento del tiempo de ejecución y a la eliminación del error de redondeo intermedio. Para el formato de vídeo 4:2:0, las muestras de crominancia se interpolan con un filtrado de 4 taps separables unidimensionales para generar 1/8 de muestra de precisión, mientras que en comparación H.264/MPEG-4 AVC sólo utiliza un filtro bilineal de 2 etapas. Al igual que en H.264, la precisión ponderada en HEVC se puede utilizar con uni-predicción (en el que se utiliza un solo valor de predicción) o bi-predicción (en el que se combinan los valores de predicción de los dos bloques de predicción).

3.2.5.6 Predicción de vector de movimiento

HEVC define un rango de 16 bits con signo para los vectores de movimiento horizontal y vertical (MV). Estos vectores de movimiento horizontales/verticales tienen un rango de -32768 a 32767, que dada la precisión de ¼ de píxel utilizado por HEVC permiten una gama MV de -8192 a 8191,75 muestras luma. Comparándolo con H.264 la mejora es más que notable puesto que los rangos son de -2048 a 2047,75.

HEVC permite dos modos diferentes de predicción de imágenes: Advanced Motion Vector Prediction (AMVP) y Merge. AMVP utiliza los datos de la imagen de referencia y también puede utilizar los datos de predicción de bloques adyacentes. El modo Merge permite a los MVs poder tomar información de bloques de predicción adyacentes. Este modo en HEVC es similar a los modos de interferencia de movimiento “saltado” y “directo” en H.264 MPEG-4 AVC teniendo en cuenta algunas mejoras:

- Utiliza información de índice para seleccionar uno de varios candidatos disponibles.
- Utiliza la información de la lista de imágenes de referencia y el índice de imagen de referencia.

3.2.5.7 Transforma inversa

HEVC especifica cuatro unidades de transformación (TUs) con tamaños de 4x4, 8x8, 16x16 y 32x32 para codificar la predicción residual. Una CTB puede ser recursiva y estar dividida en 4 o más TUs. Esta transformada utiliza TUs que, a su vez, utilizan funciones básicas integradoras muy similares a la de la transformada directa del coseno (DCT). La luminancia utiliza bloques de transformación de 4x4 pertenecientes a una región intra codificada y son transformadas mediante un número entero de transformación que es obtenido mediante la transformada sinusoidal discreta (DST). Esto proporciona una reducción de la tasa de bits del 1%, con el inconveniente de

limitar los bloques de luminancia a tamaños de 4x4. En cuanto a la crominancia utiliza los mismos tamaños que la luminancia.

3.2.5.8 Bucles de filtros

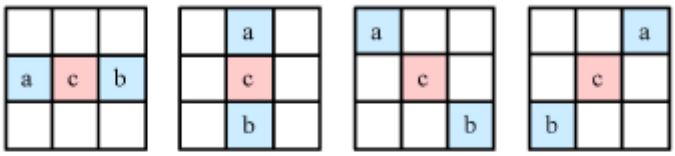
HEVC especifica dos bucles de filtros, primero se aplica el filtro *Deblocking* (DBF) y a continuación se aplica el filtro *Simple Adaptive Offset* (SAO). Ambos bucles de filtros funcionan durante el bucle de predicción inter entre imágenes. A continuación se profundizará en ambos filtros:

- Deblocking Filter: en HEVC este filtro sólo se aplica a rejillas de 8x8 píxeles, mientras que en H.264 se aplicaba a rejillas de 4x4 píxeles. Esto es debido a que mejora notablemente el procesamiento en paralelo puesto que este filtro no provoca interacciones en cascada con otras operaciones. También se requiere que el filtro se aplique primero sobre los bordes verticales de la imagen y después sobre los horizontales; esto permite que múltiples hilos paralelos se utilicen simultáneamente.

Figura 18. Efecto del filtro de "Deblocking"

- Sample Adaptive Offset: está pensado para permitir una mejor reconstrucción de las amplitudes de las señales originales mediante el uso de compensaciones de offset. A través de CTB este filtro puede ser activado o desactivado y puede funcionar en dos modos diferentes:

- *Desplazamiento del borde:*

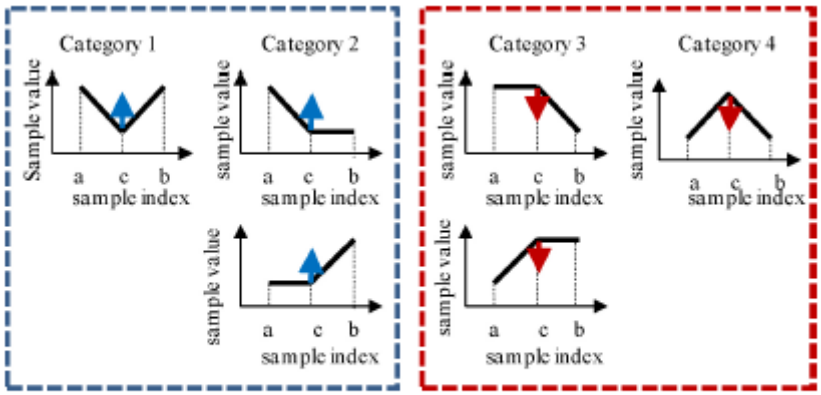


Four 1-D directional patterns for EO sample classification: horizontal (EO class = 0), vertical (EO class = 1), 135° diagonal (EO class = 2), and 45° diagonal (EO class = 3).

TABLE I

SAMPLE CLASSIFICATION RULES FOR EDGE OFFSET

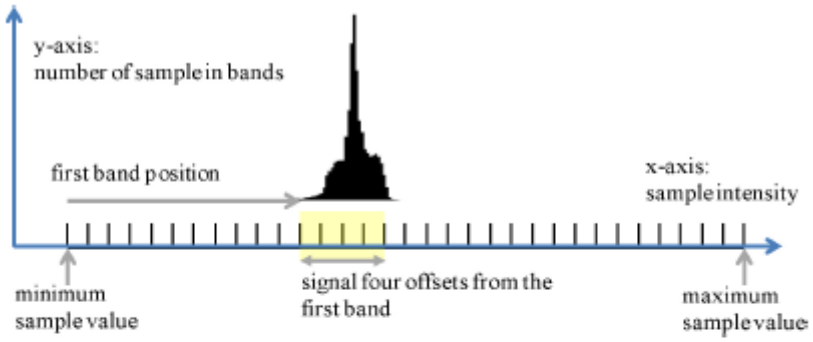
Category	Condition
1	c < a && c < b
2	(c < a && c==b) ||(c == a && c < b)
3	(c > a && c==b) ||(c == a && c > b)
4	c > a && c > b
0	None of the above



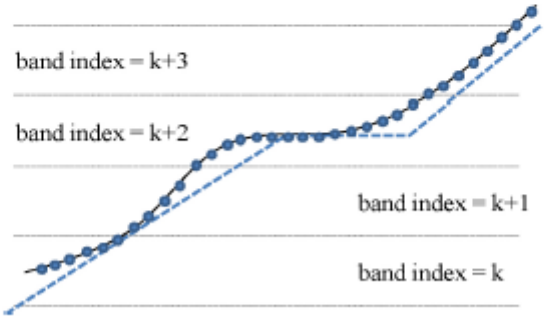
Positive offsets for EO categories 1 and 2 and negative offsets for EO categories 3 and 4 result in smoothing.

Figura 19. Ejemplo de desplazamiento de borde

- *Desplazamiento de banda*



Example of sample distribution in a CTB, where BO send the offsets of four consecutive bands.



Example of BO, where the dotted curve is the original samples and the solid curve is the reconstructed samples.

Figura 20. Ejemplo del desplazamiento de banda

Para más información sobre estos filtros consultar [REFERENCIA].

3.2.5.9 Buffer de imagen decodificada

Las imágenes previamente codificadas se almacenan en una memoria intermedia de imágenes de codificadas (DPB), y son utilizados por los decodificadores HEVC para formar predicciones para imágenes posteriores. El número máximo de imágenes que se pueden almacenar es de 6 (incluyendo la imagen actual) para todos los niveles de HEVC. Esta capacidad puede verse aumentada de 6 a 8, 12 ó 16 cuando el tamaño de las imágenes disminuye y son permitidas por el nivel. El codificador analiza las imágenes una por una y selecciona las imágenes que se almacenarán en el DPB, por lo que tiene la flexibilidad de determinar la mejor manera de utilizar la capacidad de DPB cuando se codifica el contenido del vídeo.

3.2.5.10 Perfiles

La norma HEVC define tres perfiles principales: Principal, Principal 10 y Principal imagen fija. En agosto de 2013 se definieron cinco perfiles adicionales: Principal 12, 10, Principal 4:2:2, Principal 4:2:2: 12, Main 4:4:4 10 y Main 4:4:4 12. Actualmente se están planteando extensiones futuras que incluyan un aumento de la profundidad de bits, 4:2:2/4:4:4 de crominancia, Multiview video coding (MVC) y Scalable Video Coding (SVC). Se espera que estas extensiones sean lanzadas durante el año 2014.

Un perfil es un conjunto de herramientas de codificación que se pueden utilizar para crear un flujo de bits que se ajuste a ese perfil definido. El codificador que se utilice en el perfil, puede elegir qué herramientas de codificación utilizar, siempre y cuando se genere un tren de bits adecuado; mientras que el decodificador para el perfil debe ser compatible con las herramientas de codificación disponibles en ese perfil.

Como no se va a entrar en detalles sobre los diferentes perfiles principales así como de sus extensiones, la siguiente Fig.21 resume sus características principales.

Feature	Version 1		Range extensions				
	Main	Main 10	Main 12	Main 4:2:2 10	Main 4:2:2 12	Main 4:4:4 10	Main 4:4:4 12
Bit depth	8	8 to 10	8 to 12	8 to 10	8 to 12	8 to 10	8 to 12
Chroma sampling formats	4:2:0	4:2:0	4:2:0	4:2:0/4:2:2	4:2:0/4:2:2	4:2:0/4:2:2/4:4:4	4:2:0/4:2:2 /4:4:4
4:0:0 (Monochrome)	No	No	Yes	Yes	Yes	Yes	Yes
Intra block copy	No	No	Yes	Yes	Yes	Yes	Yes
Intra smoothing disabling	No	No	Yes	Yes	Yes	Yes	Yes
Residual DPCM inter/intra	No	No	Yes	Yes	Yes	Yes	Yes
Transform skip block sizes larger than 4×4	No	No	Yes	Yes	Yes	Yes	Yes
Transform skip context/rotation	No	No	Yes	Yes	Yes	Yes	Yes
MinCR reduced to half its base value	No	No	No	Yes	Yes	Yes	Yes
Extended precision processing	No	No	No	No	No	No	No
Separate color plane	No	No	No	No	No	No	No

Figura 21. Perfiles del estándar RVC

3.2.6 Ventajas de HEVC

Las ventajas de HEVC sobre su anterior norma son muy significativas, tanto a nivel de especificaciones como de salidas en el mercado. A continuación se enumerarán las más importantes:

- Permite doblar la calidad de imagen ocupando el mismo espacio que un vídeo actual en H.264. Por lo tanto, un vídeo codificado con la misma calidad de imagen que en H.264 ocuparía la mitad de espacio.
- Ofrece una nueva ventaja para el *streaming* o para el almacenamiento en dispositivos móviles, ya que se podrá almacenar el doble de vídeos en ellos.
- Se reduce a la mitad el ancho de banda necesario para trasmitir vídeos por la Red, lo que es muy beneficioso para empresas y usuarios de telefonía.
- El códec de HEVC estará preparado para las nuevas tecnologías de vídeo 4K (3840x2160) y podría llegar a soportar 8K (7680x4320).

3.2.7 Diagrama de bloques del decodificador HEVC

Este último apartado del capítulo resume mediante la Fig.22, el diagrama de bloques de un decodificador HEVC con todas las herramientas y filtros que utiliza en el proceso de decodificación de una imagen de vídeo.

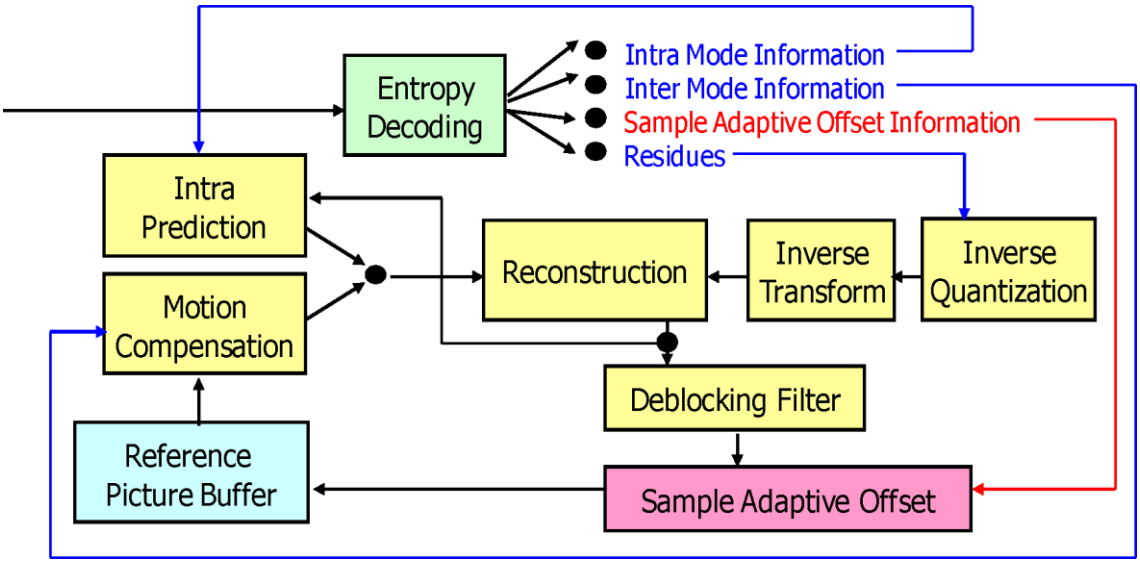


Figura 22. Decodificador HEVC

CAPÍTULO 4. Entorno de trabajo y herramientas.

En este capítulo, se comentarán todos los pasos que se han llevado a cabo para realizar la obtención del decodificador, así como de las herramientas necesarias para ello de una forma detallada.

4.1 Herramientas de trabajo

En este apartado, se explicarán todas las herramientas utilizadas tanto en el desarrollo del programa como para la obtención y ejecución del decodificador. Se analizarán independientemente cada herramienta, haciendo una descripción de la misma, así como del uso o participación que ha tenido en el desarrollo del programa.

Las herramientas que se utilizarán para llevar a cabo el proyecto serán las siguientes:

- Eclipse Indigo, tanto para generar el código de los diferentes codificadores/decodificadores, como para realizar la aplicación anteriormente comentada.

- Orcc para la compilación del código.

- Graphiti para diseñar gráficamente los decodificadores o para poder visualizar su estructura, elementos que lo componen y sus interconexiones.

- CMake para generar el código adaptado a la herramienta de compilación que se vaya a emplear.

4.1.1 Eclipse

Eclipse [**8**] es un programa informático compuesto por un conjunto de herramientas de programación de código abierto multiplataforma. Esta plataforma es usada para desarrollar entornos de desarrollo integrados (IDE), como el IDE de Java llamado *Java Development Toolkit* (JDT) y el compilador (ECJ).

Eclipse fue desarrollado originalmente por IBM, aunque ahora está siendo desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro, que fomenta una comunidad de código abierto y un conjunto de capacidades y servicios libres para los usuarios.

4.1.1.1 Rich Client Platform

La base para Eclipse es la Plataforma de cliente enriquecido (Rich Client Platform, RCP). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal: se ejecuta el inicio del programa Eclipse y se carga la ejecución de *plugins*.
- OSGi: una plataforma para agrupación (*bundling*) estándar.
- Standard Widget Toolkit: paquete de aplicaciones que ofrecen extensiones para Eclipse.
- Face: permite el manejo de archivos y textos y ejecutar editores de texto.
- Workbench: entorno de trabajo de Eclipse con vistas, editores, perspectivas y asistentes.

4.1.1.2 Características de Eclipse

En este apartado se realizará una lista con las características principales de la Plataforma de desarrollo Eclipse.

- Dispone de un editor de texto con resaltado de sintaxis, donde se puede ver el contenido del fichero sobre el que se está trabajando.
- La compilación es en tiempo real.
- Integración con Ant, asistentes (wizards) para creación de proyectos, clases, test de pruebas, etc., y refactorización.

Si bien las funciones de Eclipse son más bien de carácter general, las características del programa se pueden ampliar y mejorar mediante el uso de *plug-ins*. Asimismo, a través de estos "*plugins*" de software libre, es posible añadir un sistema de control de versiones a través de la aplicación *Subversion* y a la vez lograr una integración mediante la aplicación *Hibernate*.

4.1.1.3 Ventajas de utilizar Eclipse

Se va a realizar una enumeración de las principales ventajas de utilizar Eclipse, analizando cada una de ellas de forma individual, así como de las ventajas que poseen a la hora de crear aplicaciones cliente.

1. El entorno de desarrollo integrado (IDE) de Eclipse emplea módulos (*plug-ins*) para proporcionar toda su funcionalidad al frente de Rich Client Platform (RCP), a diferencia de otros entornos en los que las funcionalidades están todas incluidas, las necesite el usuario o no.
2. Este mecanismo de módulos es una plataforma ligera para componentes de software. Adicionalmente permite a Eclipse extenderse usando otros lenguajes de programación como son C/C++ y Phyton. Permite a Eclipse trabajar con lenguajes para procesado de texto como LaTeX, aplicaciones en red como Telnet y Sistema de gestión de base de datos.
3. La arquitectura *plug-in* permite escribir cualquier extensión deseada en el entorno, como sería Gestión de la configuración. Se provee soporte para Java y

CVS en el SDK de Eclipse. Y no tiene por qué ser usado únicamente para soportar otros lenguajes de programación.

4. La descripción o definición que da el proyecto Eclipse acerca de su Software es: "una especie de herramienta universal", ya que se trata de un IDE abierto y extensible.
5. El SDK de Eclipse incluye las herramientas de desarrollo de Java, ofreciendo un IDE con un compilador de Java interno y un modelo completo de los archivos fuente de Java. Esto permite técnicas avanzadas de refactorización y análisis de código.
6. El IDE también hace uso de un espacio de trabajo, en este caso un grupo de *metadata* en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

En el Anexo I se describen los pasos a seguir para realizar la instalación completa de Eclipse con todos sus *plugins* y herramientas adicionales.

4.1.2 Graphiti

Eclipse proporciona una infraestructura de modelado Modeling Framework (EMF), mediante representación y edición gráfica.

La aplicación Graphiti [**9**] es un editor de gráficos basada en Eclipse, que permite de forma gráfica, intuitiva y rápida el desarrollo de diagramas de bloques.

Los objetivos de la aplicación Graphiti son los siguientes:

- Proporcionar una herramienta fácil de usar y bien estructurado API Java simple para la construcción de herramientas gráficas.
- Proporcionar documentación y tutoriales para realizar los diseños gráficos.
- Proporcionar componentes opcionales más allá del caso de uso RCP para facilitar por ejemplo la integración IDE.
- Dotar de la capacidad de utilizar cualquier algoritmo de diseño existente para auto-diseñar un diagrama.

En este proyecto, la herramienta Graphiti ha sido usada para observar y comprender de manera gráfica los distintos decodificadores ya descritos, así como los bloques y actores que contienen cada uno de ellos.

La Fig.23 Presenta el nivel más alto del diseño gráfico del decodificador MPEG4 Part10 PHP. En ella se pueden apreciar el bloque Source, que proporciona la información al bloque del decodificador, así como los bloques Merger y Display.

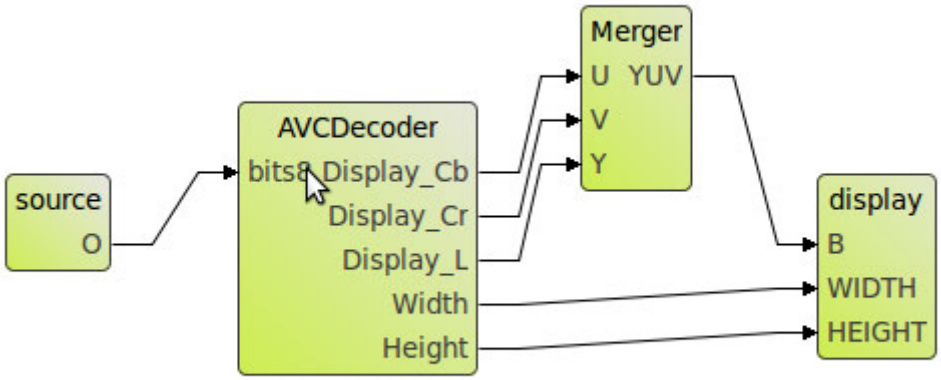


Figura 23. Representación gráfica de Graphiti del descodificador de MPEG parte 10

4.1.3 Orcc

El Open RVC-CAL Compiler (Orcc) puede generar código para distintas plataformas en los siguientes lenguajes de programación, hardware (Verilog, VHDL), software (C, Java), plataformas heterogéneas (hardware / software mixta) y las plataformas multi-softcore.

Orcc es un conjunto de herramientas de apoyo bajo la licencia *Bitstream Syntax Description* (BSD). El propósito principal de Orcc es proporcionar a los desarrolladores una infraestructura de compilador que permite varios lenguajes y la combinación de estos (en el caso de co-diseño) que se generan a partir de los actores RVC-CAL y redes xdf. Orcc no genera código ejecutable directamente, sino que genera código fuente que debe ser compilado por otra herramienta, en este caso el compilador GCC de Linux.

La siguiente Fig.24 muestra un ejemplo de lo que se entiende en Orcc como red (Network), haciendo uso de la interfaz gráfica Graphiti.

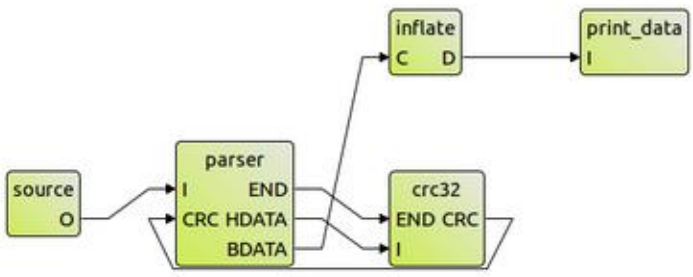


Figura 24. Network en Orcc

4.1.4 RVC-CAL

CAL (Cal Actor Language) es un lenguaje de programación de alto nivel para la descripción de los actores, que son los operadores con estado que transforman los flujos de entrada de objetos de datos (*tokens*) en corrientes de salida. CAL se ha compilado para una variedad de plataformas de destino, incluyendo los procesadores de un solo núcleo, los procesadores multinúcleo y hardware programable. Se ha utilizado en varias áreas de aplicación, incluyendo el vídeo y el procesamiento, compresión y criptografía. El Grupo de Trabajo de MPEG RVC ha adoptado CAL como parte de sus esfuerzos de estandarización.

CAL es un lenguaje de programación orientada a una variedad de dominios de aplicación, tales como el procesamiento multimedia, sistemas de control, procesamiento de red, etc. Permite disminuir considerablemente el código, permite que el diseño y la codificación de los actores sea más accesible, reduciendo la posibilidad de errores y permitiendo un código más versátil y de fácil localización de las partes que lo componen.
Gran parte del esfuerzo de programación está dirigido hacia la búsqueda de una buena factorización del problema en actores, así como de la comunicación entre los actores.

El funcionamiento básico del actor sería el siguiente: el actor recoge los *tokens* de sus puertos de entrada, a continuación se modifica el estado interno de los *tokens* recogidos y finalmente se generarían nuevos *tokens* en sus salidas.

Un ejemplo de lo que sería un actor se muestra en la Fig.25, en la que se describe el nombre y los parámetros de un actor de entrada y uno de salida. También se muestra la acción en la que el actor en el puerto de entrada pasa al puerto de salida.

```
actor <nombre>ID (<parámetros>) In==> Out: } Declaración de los puertos de E/S
    action In:[a] ==> Out: [a] end
end
```

Figura 25. Descripción de un actor

Los actores pueden tener más de una acción y pueden darse diferentes eventos o indeterminaciones en ejecución si no se programa de una manera determinada para evitarlas. Un ejemplo básico lo encontramos en la siguiente Fig.26, que muestra la declaración de dos acciones de un actor.

```
action Input1: [x] ==> [x] end
action Input2: [x] ==> [x] end
```

Figura 26. Descripción de las acciones de un actor

En los actores también se pueden indicar condiciones de guarda (*guard*), que pueden ser tanto los valores de los *tokens*, como el estado del actor o ambos. El estado de un actor puede ser utilizado tanto para el procesado como para lanzar las acciones que sean necesarias. Las operaciones que tienen que ver con el estado de los actores deben

realizarse cuando las acciones ya hayan sido ejecutadas (según la normativa de programación CAL). Un ejemplo simple sería la Fig.27:

```
actor Select() S, A, B ==> Output:
        action S: [sel], A: [V] ==>[V]
                guard sel
                 end
        action S: [sel], B: [V] ==>[V]
                guard not sel
                 end
end
```

Figura 27. Operaciones sobre el estado de un actor

En la figura se puede observar la declaración de tres actores, S, A y B y un puerto de salida. En la primera acción se indica una condición de guarda positiva, por lo que se almacena el valor de A, mientras que en la siguiente acción, la condición de guarda es negativa y por tato, no se almacena el valor de B.

Para tener una mejor visión de este lenguaje específico de programación, Se va a utilizar la Fig.28 con un ejemplo básico, un actor suma:

```
actor Suma () Input ==> Output:
    suma := 0;

    action [a] ==> [sum]
    do
        suma := sum + a;
    end
end
```

Figura 28. Descripción del actor suma en lenguaje CAL

En esta figura, se muestra la descripción de un actor Suma, al que pasándole los parámetros de entrada [a] y [sum] realiza la acción de almacenar el resultado de la suma en el parámetro de salida [suma].

4.1.5 CMake

CMake es una plataforma cruzada con sistema de compilación de código abierto. CMake [**10**] forma un conjunto de herramientas diseñadas para crear y probar paquetes de software. CMake se utiliza para controlar el proceso de compilación del software usando una plataforma sencilla y archivos de configuración independientes del compilador. CMake genera unos archivos denominados makefiles y espacios de trabajo que se pueden utilizar en el entorno del compilador propio.

CMake fue creado por Kitware en respuesta a la necesidad de un entorno

multiplataforma potente de compilación de proyectos de código abierto como ITK (*Insight Segmentation and Registration Toolkit*) y VTK (Kit de herramientas de visualización).

En este proyecto, CMake se ha utilizado, en primer lugar, para realizar el proceso de obtención del ejecutable de cada uno de los decodificadores partiendo de los ficheros obtenidos mediante la compilación con Eclipse. Este ejecutable, será el que se utilizará cuando se realice la llamada al decodificador desde el programa desarrollado.

4.1.6 GCC

La colección de compiladores de GNU incluye interfaces para C, C + +, Objective-C, Fortran, Java, Ada, y Go, así como las bibliotecas de los lenguajes de programación (libstdc + +, libgcj, ...). GCC [**11**] fue escrito originalmente como el compilador para el sistema operativo GNU. El sistema GNU fue desarrollado para ser 100% software libre.

Las fuentes de GNU son fácilmente obtenibles, por ser de libre disposición, a través de SVN instantáneas y semanales.

Este compilador viene integrado en el entorno Eclipse, por lo que no será necesaria su descarga ni instalación.

4.2 Obtención del codificador/decodificador

En primer lugar, se explicará mediante un diagrama de bloques, el procedimiento a seguir para obtener cualquiera de los decodificadores que posteriormente serán utilizados para realizar las pruebas con el programa que se desarrolla en este proyecto.

En la Fig.29 se presentará el diagrama general del flujo de trabajo:

Primeramente se debe disponer o crear y diseñar los diagramas de flujo de los actores que integran el decodificador mediante Graphiti, y también se debe poseer el código relativo a los mismos en su lenguaje de descripción propio, RVC-CAL, que contiene la descripción completa tanto de su funcionalidad como de las tareas a realizar. Una vez hecho esto, se puede generar el código correspondiente a la aplicación (decodificador) en un lenguaje de programación adecuado, en este caso C, mediante la herramienta de compilación Orcc, previamente se tienen que haber configurado correctamente las opciones de compilación teniendo en cuenta las necesidades del programador. ORCC generará un fichero con la distribución estática de la carga computacional de la secuencia de ejecución de los diferentes actores sobre cada uno de los núcleos del procesador (como es nuestro caso) trabajando en sistemas multinúcleo; este fichero tiene la extensión *.xcf

A continuación, utilizando el fichero *.bat, que se genera con anterioridad, y mediante la herramienta CMake se generan las soluciones del código generado por ORCC. CMake generará los archivos necesarios (*MakesFiles*) para posteriormente trabajar con el compilador GCC (entorno Linux) y poder generar el ejecutable del descodificador.

Una vez realizado esto, ya se dispone del ejecutable de la aplicación con el listado de actores asociados a cada núcleo del procesador. Este listado puede ser leído y modificado manualmente para realizar una nueva distribución de la carga de los actores mediante el fichero *.xcf sin necesidad de recompilar el código del descodificador. La descripción de un fichero *.xcf se realiza con más detalle en el Anexo de esta memoria.

El objetivo de este PFC es que esta distribución se realice de forma automática mediante el programa diseñado.

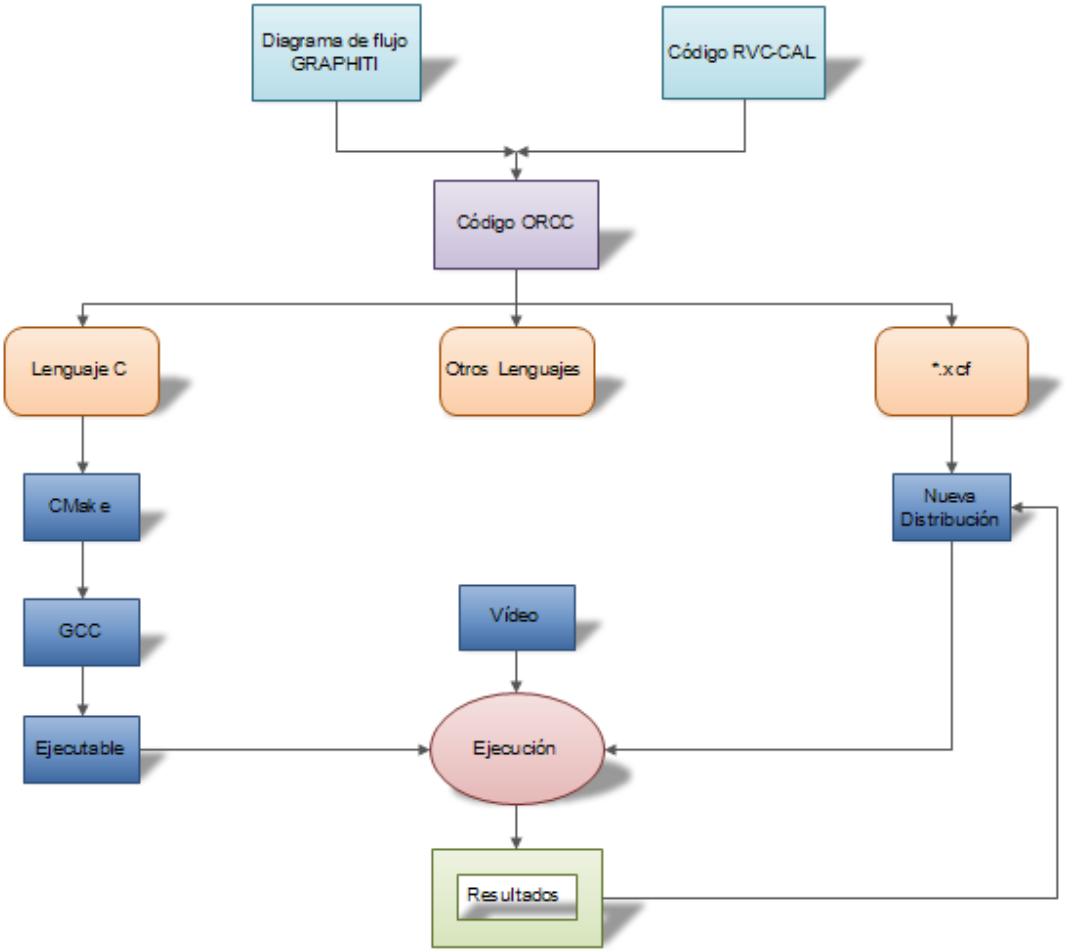


Figura 29. Diagrama de trabajo para la obtención del codificador/decodificador

CAPÍTULO 5. Aplicaciones RVC sobre plataformas multinúcleo.

Este capítulo está dedicado al estudio de las aplicaciones RVC sobre plataformas multinúcleo, de las principales características que conlleva y las ventajas que se pueden obtener con ello. Principalmente se estudiarán dos teorías, la Ley de Amdahl y la Ley de Gustafson.

También se explicarán las diferentes ventajas de emplear RVC sobre plataformas multinúcleo así como la relación entre el número de procesadores que se utilizan en relación con la ganancia que se obtiene. La ganancia hace referencia a los fps (*frames* por segundo) obtenidos en la ejecución del decodificador para una configuración de distribución de los actores en 2 o más núcleos divididos entre los fps obtenidos en la ejecución del decodificador con una configuración de distribución de 1 núcleos. En el siguiente capítulo se explica más detalladamente este concepto y la forma de obtenerlo.

5.1 Introducción a la computación paralela

La computación paralela es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo).

El paralelismo se ha empleado durante muchos años, sobre todo en la computación de altas prestaciones, pero el interés en ella ha crecido últimamente debido a las limitaciones físicas que impiden el aumento de la frecuencia de los procesadores. El consumo de energía constituye una de las grandes preocupaciones en la actualidad, por lo que la computación en paralelo se ha convertido en el principal objetivo en la arquitectura de computadores, principalmente en los procesadores multinúcleo. De forma general, cualquier programa en paralelo es más difícil de codificar que los secuenciales, ya que la concurrencia introduce más parámetros a tener en cuenta y la aparición de nuevos errores de software. La comunicación y sincronización entre diferentes subtareas son algunos de los mayores obstáculos para obtener un buen rendimiento del programa paralelo.

La máxima aceleración posible de un programa como resultado de la paralelización se conoce como la ley de Amdahl, que se explicará a continuación.

5.1.1 Ley de Amdahl

Idealmente, la aceleración a partir de la paralelización es lineal, doblar el número de elementos de procesamiento debe reducir a la mitad el tiempo de ejecución y doblarlo

por segunda vez debe nuevamente reducir el tiempo a la mitad. Sin embargo, muy pocos algoritmos paralelos logran una aceleración óptima. La mayoría tienen una aceleración casi lineal para un pequeño número de elementos de procesamiento, y pasa a ser constante para un gran número de elementos de procesamiento.

La aceleración potencial de un algoritmo en una plataforma de cómputo en paralelo está dada por la ley de Amdahl, formulada originalmente por Gene Amdahl en la década de 1960. Esta señala que una pequeña porción del programa que no pueda paralelizarse va a limitar la aceleración que se logra con la paralelización. Los programas que resuelven problemas matemáticos o de ingeniería, típicamente consisten en varias partes paralelizables y varias secuenciales. Si α es la fracción de tiempo que un programa gasta en partes no paralelizables, luego:

$$S = \frac{1}{\alpha} \qquad\qquad S = \lim_{p \to \infty} \frac{1}{\frac{1-\alpha}{p} + \alpha}$$

es la máxima aceleración que se puede alcanzar con la paralelización del programa. Esto pone un límite superior a la utilidad de añadir más unidades de ejecución paralelas. Cuando una tarea no puede divididirse debido a las limitaciones secuenciales, la aplicación de un mayor esfuerzo no tiene efecto sobre la programación.

La Fig.30 muestra un ejemplo de la relación entre el número de procesadores y la aceleración (velocidad de ejecución).

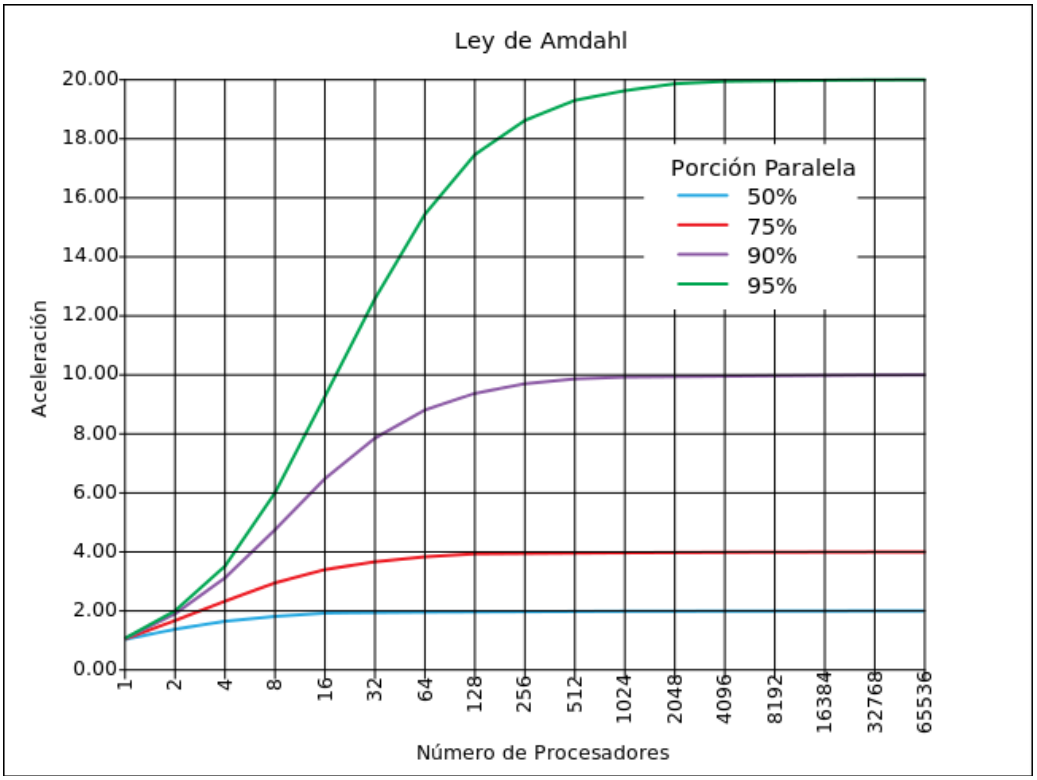


Figura 30. Ley de Amdahl

La mejora en la velocidad de ejecución de un programa como resultado de la paralelización está limitada por la porción del programa que no se puede paralelizar. Por ejemplo, si el 10% del programa no puede paralelizarse, el máximo teórico de aceleración utilizando la computación en paralelo sería de 10x no importa cuántos procesadores se utilicen.

Para más información sobre la Ley de Amdahl consultar referencia [**12**].

5.1.2 Ley de Gustafson.

La Ley Gustafson establece que cualquier problema suficientemente grande puede ser paralelizado eficientemente. Esta ley aborda las limitaciones que puede tener la ley de Amdahl a la hora de aumentar el número de máquinas o procesadores en ejecución. Gustafson propone que el programador establezca el tamaño de los problemas que va a abordar para utilizar las herramientas disponibles en su solución en un tiempo práctico. Por lo tanto, si se tienen herramientas más eficientes, mayores problemas podrán ser abordados en el mismo tiempo.

Los cambios realizados respecto a la Ley de Amdahl son los siguientes:
El tiempo de ejecución de un programa en una plataforma paralela se desglosa en:
$(a+b)$
Donde a es el tiempo secuencial y b el paralelo en cualquiera de los P procesadores.
La teoría de esta Ley es que la cantidad de trabajo que se realiza en paralelo varía de forma lineal con el número de procesadores utilizados.
El tiempo correspondiente para el procesamiento secuencial es:
$t = a + P \cdot b$
El aceleramiento (*speedup*) se realiza en concordancia mediante:

$$speedup = \frac{(a + P \cdot b)}{(a + b)}$$

Y definiendo α como la fracción secuencial del tiempo de ejecución en paralelo

$$\alpha = \frac{a}{(a+b)}$$

Se obtiene finalmente la ley de Gustafson:

$$S(P) = \alpha + P \cdot (1 - \alpha) = P - \alpha \cdot (P - 1)$$

La siguiente Fig.31 ilustra el comportamiento de la Ley de Gustafson.

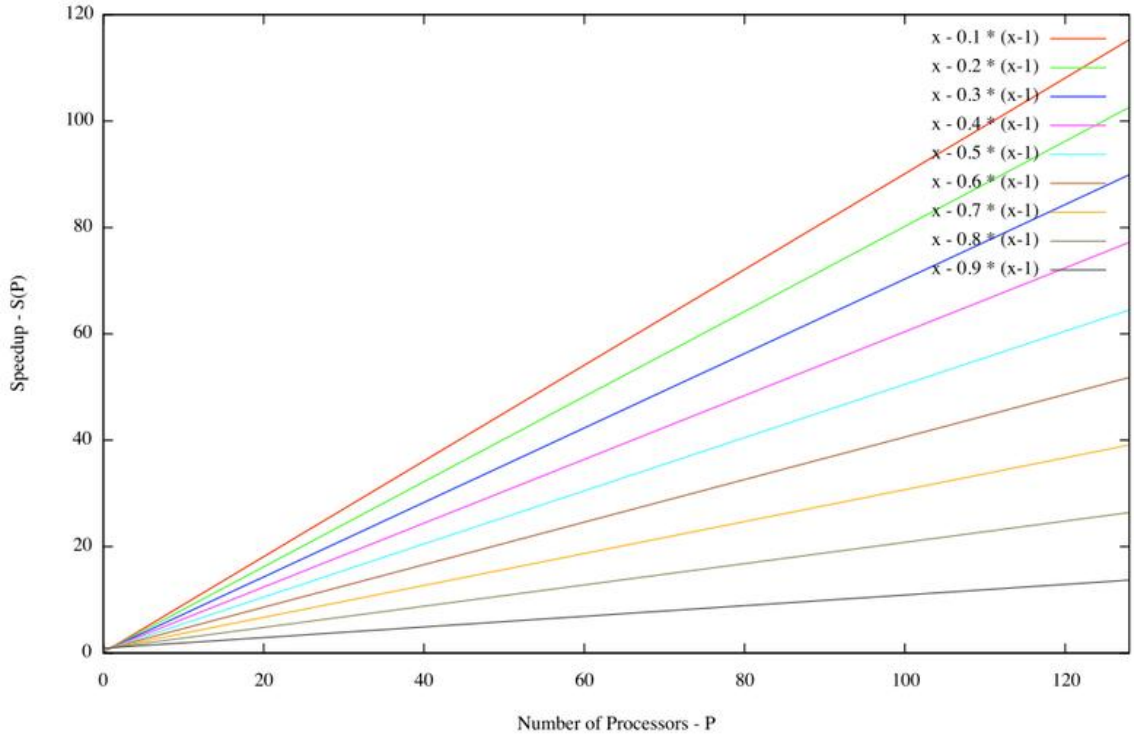


Figura 31. Ley Gustafson

Si α es pequeño, el aceleramiento es aproximadamente P, como es deseado. Incluso se puede dar el caso en que α disminuya mientras P junto con el tamaño del problema aumente; si esto se cumple, S se aproxima a P monótonamente con el crecimiento de P.

5.1.3 Paralelismo escalable y Modularidad

La computación paralela es una forma de cómputo en la que muchas instrucciones se ejecutan simultáneamente, operando sobre el principio de que problemas grandes, a menudo se pueden dividir en unos más pequeños, que luego son resueltos simultáneamente (en paralelo).
La creación de programas de tal manera que sus partes se ejecutan concurrentemente sin mucha interferencia es uno de los problemas clave en la escala tradicional de programas imperativos. Los actores encapsulados permiten exponer más el paralelismo cuando las aplicaciones crecen en tamaño.

La fuerte encapsulación de los actores junto con su estructura jerárquica ofrece un alto grado de modularidad. Por lo tanto, la especificación interna de cualquier actor puede ser modificado sin afectar a otros actores.

5.1.4 Flexibilidad

A diferencia de los lenguajes de programación de procedimiento, en los que el control de flujo (secuencia de ejecución de instrucciones) está estrechamente ligada con el algoritmo, el modelo de actor permite más flexibilidad en el proceso de programación del modelo (es decir, determinar el orden de ejecución de las acciones) por lo que permite varios esquemas de programación teniendo en cuenta diferentes criterios de optimización.

5.1.5 Portabilidad

Para muchos programas altamente concurrentes, la portabilidad siempre ha sido un objetivo difícil de alcanzar, debido a su sensibilidad a la sincronización. La pérdida de tiempo y la asincronía son una solución de programación de flujo de datos para este problema. Al proporcionar estas características, CAL es claramente un modelo flujo de datos adecuado modelo de programación para el modelado y la implementación de complejos los sistemas de procesamiento de señales en las plataformas multinúcleo.

5.2 Ventajas de RVC en plataformas multinúcleo

RVC y los estándares de vídeo están pensados para trabajar sobre plataformas multinúcleo, debido a los rápidos avances tecnológicos en el ámbito de los microprocesadores. Esto, ofrece una serie de ventajas [**14**] que se detallarán a continuación:

- Si se trabajase sobre una plataforma de un solo núcleo, toda la carga computacional recaería sobre él. Al trabajar con plataformas de más de un núcleo, dicha carga se reparte en los diferentes núcleos, obteniendo una ejecución más rápida y eficiente.
- Se pueden ejecutar simultáneamente varios procesos, repartidos en los diferentes núcleos, con lo que se disminuye el tiempo de ejecución.
- A la hora de realizar la asignación de los actores de codificación de vídeo sobre los diferentes núcleos, se tiene prácticamente una libertad completa para realizar la distribución y obtener así el mayor rendimiento posible, obteniendo los mejores valores de ganancia respecto a la distribución en un núcleo.

CAPÍTULO 6. Desarrollo del programa.

Hay que tener en cuenta que el resultado de realizar el diseño de un decodificador mediante RVC es la obtención de un código fuente (en diferentes lenguajes de programación como C/C++, Java, VHDL u otros) que puede ser ejecutado en diversas plataformas. Una de las opciones más interesantes es la ejecución del decodificador en varios procesadores de forma simultánea (sistemas multiprocesador), en los que la carga computacional se puede dividir entre los procesadores del sistema.

Hasta la fecha, este reparto de los actores del decodificador se realiza de forma manual por parte del diseñador. Esta tarea es larga y costosa e implica tener que disponer de un conocimiento profundo de los módulos que componen el sistema desarrollado.

Este proyecto está enfocado al desarrollo de una herramienta que permita realizar la asignación de cada uno de los actores del descodificador de forma automática y eficiente, almacenando las mejores distribuciones obtenidas a partir de los datos de rendimiento que se obtienen con cada uno de ellos.

Para poder llegar a desarrollar completamente este programa, se han llevado diferentes tareas o fases que se explicarán a continuación:

- Estudio de la codificación de vídeo: adquisición de conocimientos de forma genérica de la descripción y composición de imágenes y vídeos así como la forma de codificación de los mismos.
- Estudio de RVC: análisis de los diferentes estándares de codificación de vídeo (H264 y H265) así como de los decodificadores utilizados, MPEG 4 Parte 2, MPEG 4 Parte 10 CBP y PHP y por último HEVC.

Estos puntos ya han sido desarrollados en los capítulos 2, 3 y 4.

- Prueba iniciales con los diferentes decodificadores y estándares de codificación para entender y comprender cada uno de los actores que componen cada uno de los decodificadores.
- Desarrollo de la aplicación, tanto a nivel de usuario como a nivel de codificación interna.

Estos puntos serán analizados y desarrollados en profundidad a lo largo de este capítulo.

- Pruebas de funcionamiento, que serán analizadas en el siguiente capítulo.
- Obtención de un banco de pruebas realizadas para analizar el comportamiento de la aplicación desarrollada.

Este capítulo se va a dividir en 2 principales apartados diferenciados: un apartado en el que se desarrollará la interfaz de usuario y otro en la que se analizará el desarrollo interno del programa.

En primer lugar, hay que especificar que el programa se ha desarrollado sobre una plataforma Linux, mediante el programa Eclipse y en lenguaje de programación C.

6.1 Interfaz de usuario

En este apartado se va a desarrollar los pasos a seguir por el usuario para el correcto funcionamiento del programa. Para entender mejor estos pasos (interfaz de usuario) se utilizará el diagrama de bloques de la siguiente Fig.32

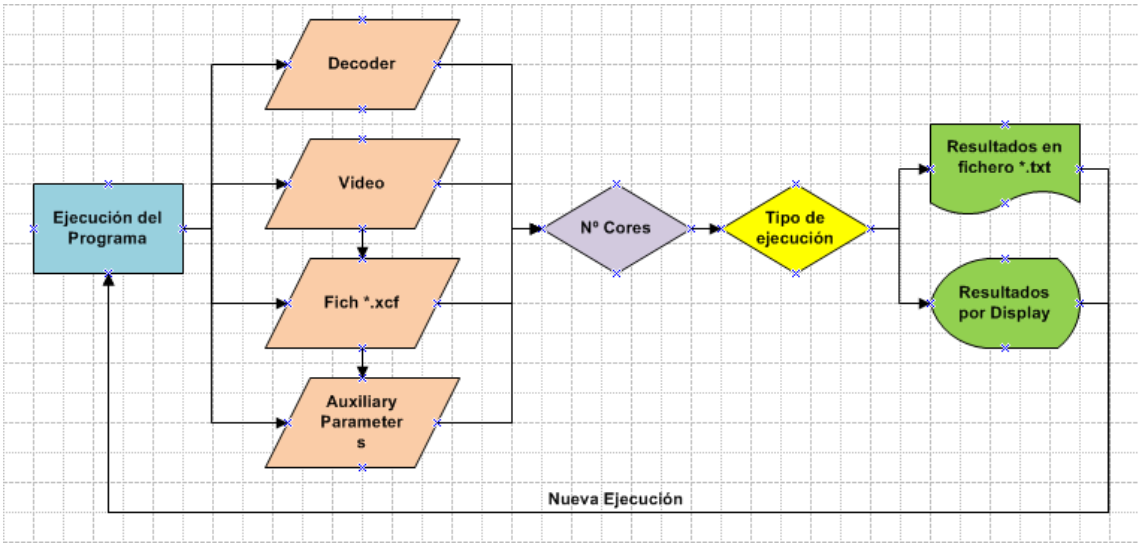


Figura 32. Interfaz de usuario

6.1.1 Ejecución del programa

En primer lugar, el usuario tiene que realizar la ejecución del programa que se ha desarrollado para poder comenzar a utilizar la aplicación. Hay dos formas de lanzar la ejecución del programa: mediante el terminal o mediante el programa eclipse. En ambas ejecuciones, los pasos a seguir son los mismos, por lo que sólo se explicará el proceso mediante la ejecución en el terminal.

Desde el terminal se ejecuta el programa desarrollado, que estará situado en el espacio de trabajo, en la carpeta *Debug* con el nombre del programa. Un ejemplo de esta ejecución sería la siguiente Fig.33

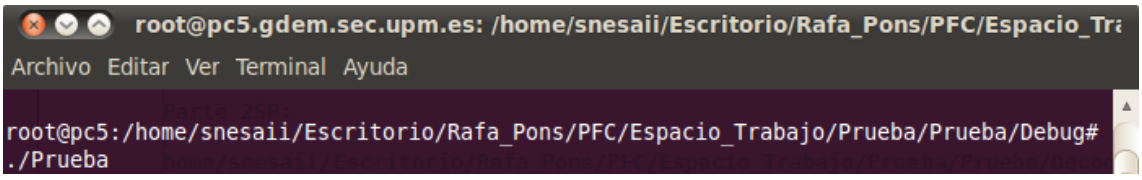


Figura 33. Ejecución del programa mediante el terminal de Linux

6.1.2 Selección de parámetros

Una vez realizada la ejecución del programa, éste pedirá al usuario introducir los parámetros básicos de configuración de la ejecución así como los parámetros auxiliares de ejecución (si el usuario lo ve conveniente).

Los parámetros básicos para la ejecución del programa son:

- Decodificador: establece el decodificador que se va a utilizar para decodificar el vídeo.
- Vídeo: se elige el fichero de vídeo a decodificar.
- Fichero de distribución *.xcf con la asignación de todos los actores a un solo núcleo.

Para introducir estos parámetros, se hará a través del terminal e indicando la ruta completa de cada uno de ellos. La estructura sería la siguiente:

Path_Decodificador –i path_vídeo –m path_fichero.xcf

A continuación de los parámetros anteriores, se permite introducir también parámetros auxiliares con diferentes funcionalidades:

> ***-n***: para evitar la visualización del vídeo por pantalla. Si este parámetro no es introducido, se establecerá la configuración por defecto, que es la visualización del vídeo.
> ***-l "nº"*** para establecer el número de veces que se repite la ejecución del vídeo. La opción por defecto es la repetición indefinida de la decodificación del vídeo, por lo que el usuario deberá introducir algún valor para el correcto funcionamiento del programa desarrollado. La repetición del vídeo permite que se tomen sucesivos datos de los fps obtenidos en la ejecución para finalmente realizar la media aritmética de estos valores. Esto permite una mayor precisión a la hora de tomar los resultados.
> ***> "Nombre del fichero de salida"***: para que los resultados de la ejecución que se muestran en el terminal se almacenen en un fichero tipo texto, en vez de mostrarse por pantalla, que es la opción por defecto.

Nota: es necesario introducir en primer lugar los parámetros obligatorios y a continuación los parámetros opcionales si el usuario lo ve conveniente o necesario.

A continuación se mostrará un ejemplo de la forma de ejecutar el decodificador con la introducción de los parámetros. Mediante la Fig.34 se ilustrará este proceso desde el terminal de Linux, mientras que la Fig.35 lo hará desde el programa Eclipse.
Un ejemplo de la estructura completa de los parámetros de ejecución es:

Path_Decodificador –i path_vídeo –m path_fichero.xcf -n -l 5 > Fichero_Salida

```
Ejecutando el programa...
Introduzca los datos para la ejecucion del decodificador:
(Parámetros obligatorios: Decodificador, video, fichero.xcf de 1 núcleo)
/home/snesaii/Escritorio/Rafa_Pons/PFC/Espacio_Trabajo/Prueba/Prueba/Decodificadores/
Top_mpeg4_part10_CBP_decoder -i /home/snesaii/Escritorio/Rafa_Pons/PFC/Espacio_Trabaj
o/Prueba/Prueba/Videos/Part10_CBP/HCBP2_HHI_A.264 -m /home/snesaii/Escritorio/Rafa_Po
ns/PFC/Espacio_Trabajo/Prueba/Prueba/Distribuciones/Part10_CBP/1Core.xcf -n -l 1
```

Figura 34. Introducción de parámetros de ejecución desde terminal de Linux

Los parámetros introducidos en el terminal son: decodificador, fichero de vídeo, fichero de distribución, visualización del vídeo y número de repeticiones del vídeo

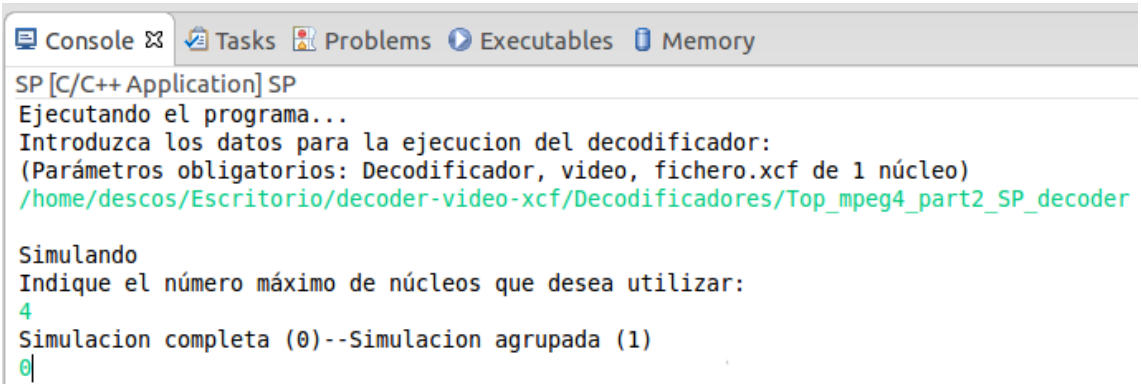


Figura 35. Introducción de parámetros de configuración desde Eclipse

6.1.3 Número de procesadores y tipo de simulación

Una vez se haya realizado la simulación, que se ejecuta automáticamente al introducir los parámetros mencionados en el punto anterior, se tendrá que introducir el número de núcleos sobre los que trabajará el programa así como si se desea una simulación completa o agrupada.

La elección del número de núcleos hace referencia al número de núcleos/procesadores que podrá utilizar el programa para realizar todas las distribuciones posibles de los actores sobre dichos núcleos.

Nota: El ordenador en el que se ejecute el programa, debe tener, al menos, los mismos procesadores que el número de núcleos que el usuario establezca para realizar las simulaciones.

La elección del tipo de simulación que el usuario puede hacer es la siguiente: una simulación completa, en la que el programa realiza el total de las distribuciones posibles; y una simulación agrupada, en la que se realizan un menor número de simulaciones ya que los actores de menor carga se agrupan en un mismo bloque de

actores. Todo esto se explicará detalladamente en el subcapítulo del desarrollo interno del programa.

La Fig.36 ilustra esta parte de la ejecución:

```
Simulando
Indique el número máximo de núcleos que desea utilizar:
2
Simulacion completa (0)--Simulacion agrupada (1)
0

End of simulation !
```

Figura 36. Número de núcleos y tipo de simulación

6.1.4 Obtención de resultados

Una vez se ha realizado la ejecución del programa, introducido los parámetros, seleccionado el número de núcleos a utilizar y el tipo de simulación, el usuario simplemente tendrá que esperar a que el programa realice automáticamente todas las distribuciones posibles, y por lo tanto, todas las simulaciones posibles para obtener los resultados.

Los resultados pueden ser mostrados por pantalla como se muestra en la siguiente Fig. 37 o en un fichero de texto.

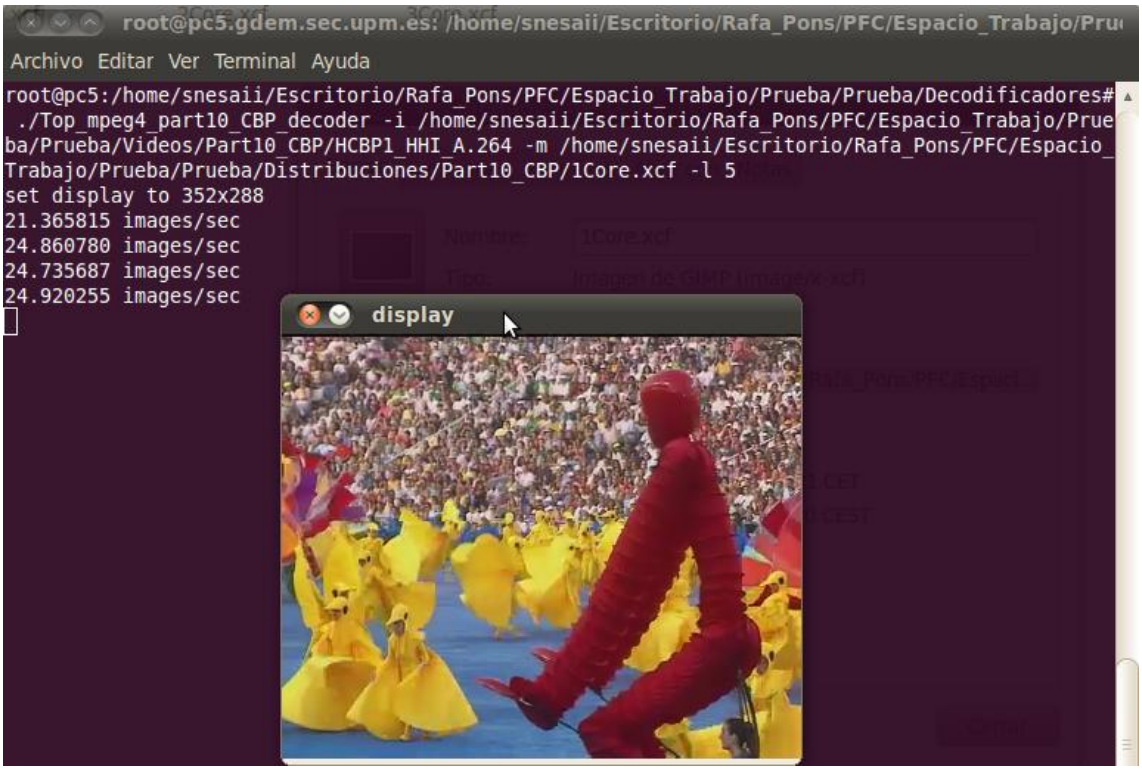


Figura 37. Obtención de resultados desde el terminal

El usuario puede encontrar estos resultados guardados en un fichero tipo texto con el nombre “Resultados.txt”, que se encuentra en la carpeta del proyecto de trabajo. Este fichero contendrá las mejores distribuciones obtenidas en la ejecución del programa junto con los datos de las imágenes, tiempo, frames por segundo y ganancia obtenidas para cada distribución. El formato y el contenido de este fichero se explicarán más en profundidad en el subcapítulo 6.2.9.

6.2 Desarrollo interno del programa

A continuación se describirá todo el proceso y desarrollo de ejecución interna del programa creado para este proyecto. El siguiente diagrama de la Fig.38 permitirá entender de una manera más cómoda y sencilla el desarrollo del mismo así como las fases de ejecución del programa.

En primer lugar, se ejecuta el decodificador pidiendo al usuario final la introducción de los parámetros básicos: decodificador, fichero de vídeo y fichero *.xcf para un núcleo. A continuación se obtienen dichos parámetros de ejecución y se almacenan en variables independientes para su posterior utilización. Se recorre y analiza cada uno de los actores del fichero de distribución para obtener los Bloques de actores del decodificador utilizado. Se obtienen los resultados de la simulación; si los resultados superan un mínimo de ganancia establecido, dichos resultados así como la distribución, son almacenados en un fichero de texto. Si por el contrario no supera este límite, se realiza una nueva distribución de los actores y una nueva ejecución con dicha distribución.

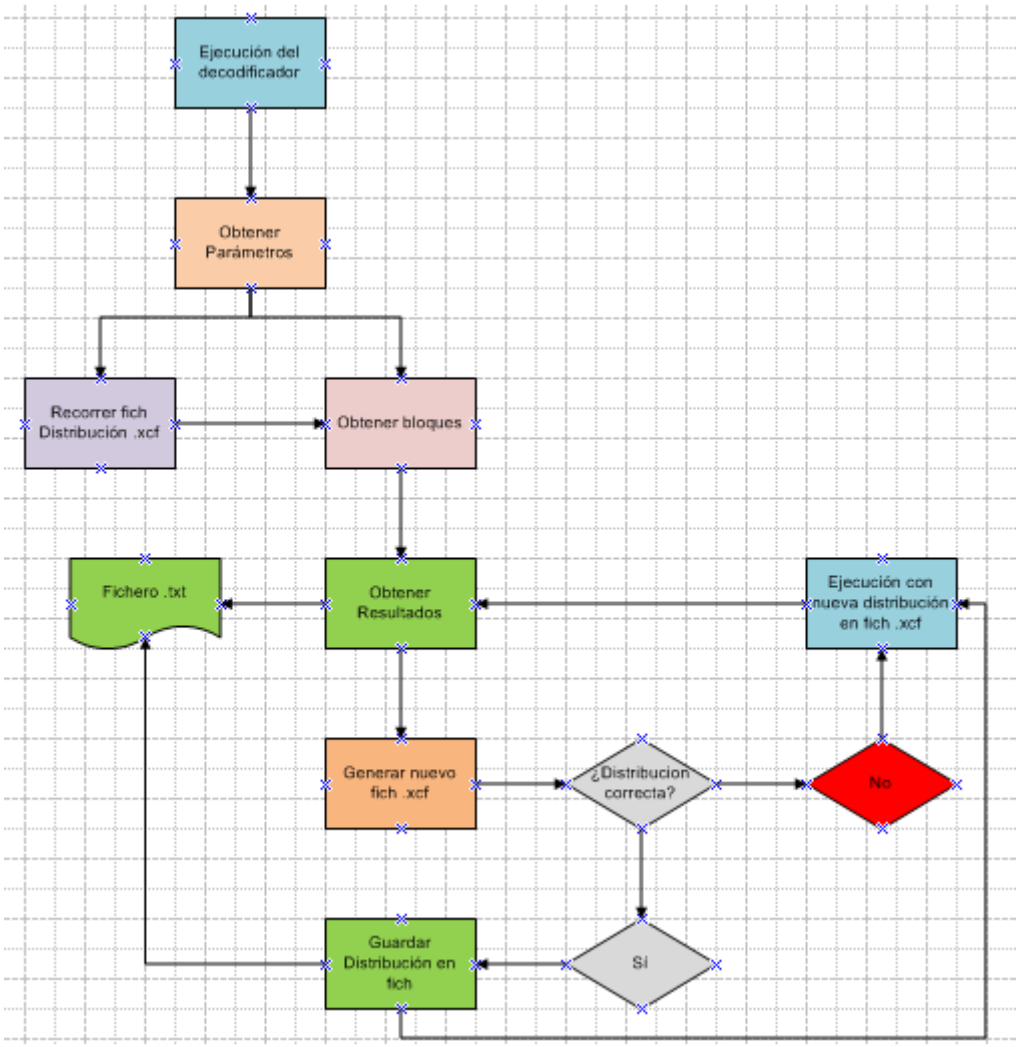


Figura 38. Desarrollo del programa

6.2.1 Ejecución del decodificador

Lo primero que realiza el programa una vez que es ejecutado por el usuario es la ejecución del decodificador con la distribución de los actores sobre un núcleo. Para ello, se pide al usuario que introduzca por teclado unos parámetros básicos (decodificador, video y fichero *.xcf de un núcleo), así como el número de núcleos y tipo de simulación.
El prototipo de la función es el siguiente:

```
int Ejecutar_Deco(char *path_completo, TParametros Parametros);
```

Donde path_completo es la variable char* donde se almacenan todos los parámetros introducidos por el usuario mediante teclado para su ejecución; y Parámetros es la variable tipo estructura en la que se guardan los valores de cada uno de los parámetros principales y auxiliares introducidos por el usuario.

En las sucesivas llamadas a la aplicación desarrollada, que se realizarán de forma automática con los parámetros introducidos en la ejecución inicial, se mantendrán todos los parámetros exceptuando el fichero de distribución *.xcf, que será modificado automáticamente por el programa para buscar el mejor rendimiento posible.

6.2.2 Obtención de parámetros

Una vez realizada la ejecución del decodificador, se obtienen todos los parámetros introducidos por teclado por el usuario. La manera elegida para llevar a cabo esta tarea es leer carácter a carácter la variable `char path`, en la que se almacenan todos los parámetros introducidos por el usuario de forma continua, y separar cada parámetro almacenándolos en una estructura con la siguiente forma:

```
typedef struct
{
	char *path_deco;
	char *path_aux_video, *path_video;
	char *path_aux_xcf, *path_xcf, *path_xcf_new;
	char *path_visualizar_video;
	char *path_aux_loop, *path_loop;
	char *path_resto;
	char *path_aux_fich_salida, *path_fich_salida;
}TParametros;
```

A continuación se explica qué almacena cada una de las variables de la estructura:

`char *path_deco`: ruta completa del decodificador.

`char *path_aux_video`: parámetro auxiliar *–i* que indica que el parámetro que se escribe a continuación es la ruta del vídeo.

`char *path_video`: ruta completa del vídeo.

`char *path_aux_xcf`: parámetro auxiliar *-m* que indica que el parámetro que se escribe a continuación es la ruta completa del fichero de distribución *.xcf.

`char *path_xcf` y `char *path_xcf_new`: ruta completa del fichero de distribución introducido por el usuario y el generado por el programa para las sucesivas simulaciones respectivamente.

`char *path_visualizar_video`: parámetro auxiliar *–n* que indica que durante la ejecución no se muestre la imagen del vídeo que se está decodificando; si este parámetro no se pusiese, en cada ejecución se mostraría el vídeo.

`char *path_aux_loop`: parámetro auxiliar *-l* que indica que el parámetro que se escribe a continuación es el número de veces que se repite la ejecución del vídeo con cada distribución realizada.

`char *path_loop`: número de veces que se repite la ejecución del vídeo.

`char *path_resto`: resto de parámetros auxiliares que introduzca el usuario.

`char *path_aux_fich_salida`: ruta completa del fichero de salida donde se almacenan los resultados de cada simulación.

`char *path_fich_salida`: ruta completa del fichero final de salida.

Esta fase se realiza inmediatamente después de realizar la ejecución del decodificador para obtener todos los parámetros introducidos. Estos parámetros serán posteriormente utilizados para llevar a cabo cada ejecución del programa, sin necesidad de que el usuario tenga que introducirlos nuevamente.

El prototipo de la función es:

```
void ObtenerParametros (char *path_completo, TParametros *Parametros);
```

Donde path_completo es la variable donde se almacenan de forma lineal todos los parámetros para llevar a cabo las sucesivas ejecuciones; y Parametros corresponde a la dirección de memoria donde están almacenados dichos parámetros.

6.2.3. Recorrer el fichero de distribución y obtención de bloques funcionales

Antes de entrar en detalle sobre el proceso recorrer el fichero de distribución y la obtención de bloques, se va a aclarar la diferencia entre un actor del decodificador y un bloque de actores.

Los actores de un decodificador son los operadores con estado que transforman los flujos de entrada de datos (tokens) en corrientes de salida. Dependiendo del decodificador que se utilice varían tanto el número de actores como la descripción de los mismos.

Lo que se denomina Bloque de actores, es el conjunto de actores que comparten una funcionalidad común, y que son agrupados para formar un bloque. Un ejemplo de un bloque de actores se puede encontrar en la siguiente página.

La descripción de lo que es un fichero de distribución de los actores del decodificador *.xcf y su contenido se detalla en el Anexo 2. En este momento se recomienda revisar este anexo para entender mejor esta parte del programa.

Una vez realizada la aclaración, se entra en detalle en esta fase de ejecución del programa. En esta parte del programa se lee completamente el fichero de distribución *.xcf (que contiene todos los actores del decodificador) y se obtienen todos los bloques que componen el fichero con los actores correspondientes a cada uno de los bloques. Tras esta fase, se dispone de estructuras con todos los bloques y actores del decodificador. En el Anexo 2 se pueden encontrar todos los actores de un decodificador y los bloques de actores que forman.

El proceso que lleva a cabo esta función es el siguiente:
Se compara cada línea leída con la anterior; si coinciden los primeros “n” parámetros, indica que la línea leída pertenece al mismo bloque de actores que la línea anterior. En

el caso de que no hubiese coincidencia, se crearía el siguiente bloque. Esta comparación se realiza en un bucle continuo hasta que se haya leído todo el fichero de distribución.

La estructura de los bloques es:

```
typedef struct
{
	char contenido[TAM_MAX_BLOQUE];
	double gain_bloque;
	int nucleo;
	int flag_bloque;
	int flag_bloqueaux;
}TBloques;

typedef struct
{
	TBloques bloques[MAX_BLOQUES];
}TBloqueGeneral;
```

La estructura *TBloques* contiene la ganancia de cada bloque, el núcleo al que es asignado cada uno de los bloques, mediante los flag se indica si el bloque pertenecerá a un solo bloque o será desplazado al bloque agrupado (en el que se almacenan los bloques con menor carga computacional en la simulación agrupada); mientras que la estructura *TBloqueGeneral* contiene todos los bloques que se crean durante la ejecución.

La ganancia de cada uno de los bloques de actores obtenidos, se calcula de la siguiente forma: se crea un fichero de distribución *.xcf dejando un bloque de actores en un núcleo, y el resto de bloques en otro núcleo. Se ejecuta el decodificador y se realiza el cálculo de la ganancia del bloque con los fps obtenidos divididos entre los fps con la configuración de distribución de un núcleo. Este proceso se repite para cada uno de los bloques. El bloque con la menor ganancia es el que se utiliza como ganancia mínima de referencia; a su vez, se establece un margen de un 15% para establecer el umbral. Si la ganancia de un bloque no supera la ganancia mínima de referencia + un 15%, se considera que el bloque tiene poca carga computacional y se crea un nuevo bloque de bloques de actores con aquellos que no superen el umbral, generando el denominado Bloque Agrupado.

En el programa se han distinguido dos tipos de bloques: el Bloque Fijo y el Bloque.
El Bloque Fijo hace referencia al contenido que no varía del fichero de distribución *.xcf.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
	<Partitioning>
		</Partition>
	</Partitioning>
</Configuration>
```

El Bloque hace referencia al conjunto de actores que forman cada uno de los bloques, un ejemplo sería el conjunto de actores que forman el Bloque Parser:

```
<Instance id="AVCDecoder_Syn_Parser_Algo_Synp"/>
<Instance id="AVCDecoder_Syn_Parser_Algo_Parser_IPCM"/>
<Instance id="AVCDecoder_Syn_Parser_IntraPredSplit"/>
<Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MMCO"/>
<Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_RefList0"/>
<Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_Mgnt_InterPred"/>
<Instance
id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MgntDeblockingFilter"/>
<Instance
id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvL0_Reconstr"/>
<Instance
id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_RefIdxL0ToFrameNu
m"/>
<Instance
id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvComponentReord"
/>
```

Mediante una de las partes significativas del código de la función se puede entender mejor el proceso de obtención de los bloques.

```
void ObtenerBloques(char *nombre_Archivo, char BloqueAux[TAM_BLOQUE_FIJO],
TBloqueGeneral *BloqueGeneral)
{
	static FILE *fich_distribucion = NULL;
	char linea[LONG_MAX_LINEA], linea_ant[LONG_MAX_LINEA];
	bloques=0;

	if ((fich_distribucion = fopen(nombre_Archivo, "r")) == NULL
		perror(nombre_Archivo);

	while (fgets(linea, LONG_MAX_LINEA, fich_distribucion))
	{
		if ((strstr(linea,"version")!=NULL) ||
		(strstr(linea,"Configuration")!=NULL) ||
		(strstr(linea,"Partitioning")!=NULL))
			strcat (BloqueAux, linea);
		else if (strstr(linea, "Partition id=")!=NULL)
		{
		}
		else if ((strstr(linea_ant, "Partition id=")!=NULL) ||
		(strncmp(linea, BloqueGeneral->bloques[0].contenido,
		MAX_COMPARACION)==0))
			strcat (BloqueGeneral->bloques[0].contenido, linea);
		else if (!BloqueGeneral->bloques[1].flag_bloque &&
		BloqueGeneral->bloques[0].contenido != NULL)
		{
			strcat (BloqueGeneral->bloques[1].contenido, linea);
			BloqueGeneral->bloques[1].flag_bloque = 1;
		}
		else if (strncmp(linea, BloqueGeneral->bloques[1].contenido,
		MAX_COMPARACION)==0)
```

```
                strcat (BloqueGeneral->bloques[1].contenido, linea);
        else if (!BloqueGeneral->bloques[2].flag_bloque &&
        BloqueGeneral->bloques[1].contenido != NULL)
        {
                strcat (BloqueGeneral->bloques[2].contenido, linea);
                BloqueGeneral->bloques[2].flag_bloque = 1;
        }
...
...
    while (bloques < MAX_BLOQUES)
    {
            printf("Bloque %d:\n %s\n", bloques, BloqueGeneral-
            >bloques[bloques].contenido);
            if (BloqueGeneral->bloques[bloques].contenido != NULL)
                    bloques ++;
    }
    fclose(fich_distribucion);
```

6.2.4. Obtención de resultados

Cuando se realiza la ejecución del decodificador, y se realiza cualquiera de las simulaciones, los resultados del tamaño del vídeo, número de imágenes/seg y frames por segundo que se obtienen se almacenan en un fichero previamente generado en el apartado de ejecución del decodificador (6.2.1), estableciendo de forma implícita el parámetro > *Fich_Out*. Un ejemplo del fichero generado sería el de la Fig.39:

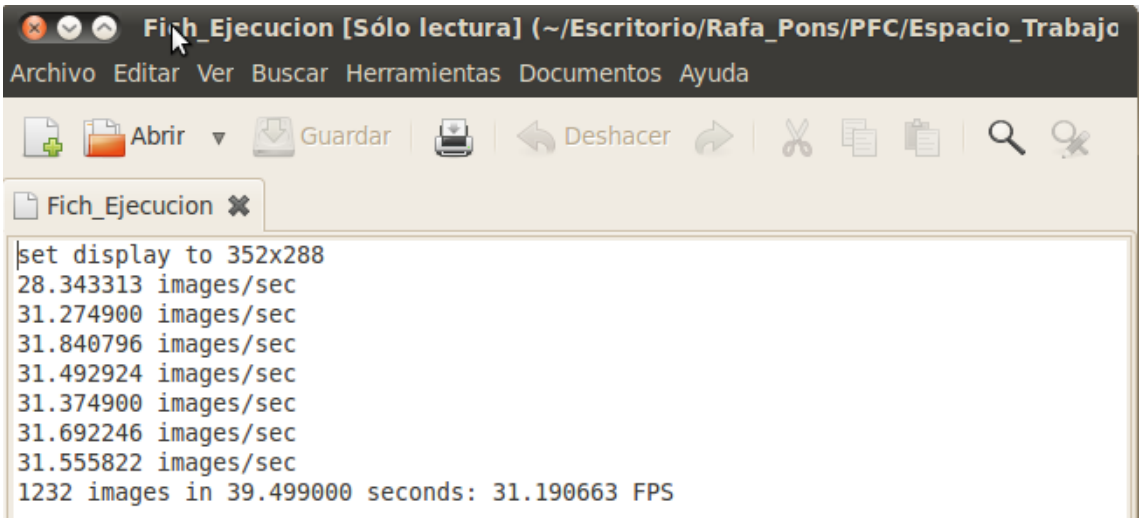


Figura 39. Resultados de ejecución almacenado en fichero del programa

Para obtener los datos, se lee el fichero que se crea en el espacio de trabajo y se observa que hay tres datos relevantes: *images*, *time* y *fps* (frames por segundo). Como los datos están escritos es un fichero, tienen tipo char, por lo que, mediante la función atoi y atof se convierten los datos en tipo int y float respectivamente.

Una vez convertidos los datos, se pasa a la obtención del rendimiento del programa calculando la ganancia obtenida en la ejecución. Para ello, se realizará el siguiente cálculo:

```
gain = fps/fps_ini;
```

`double fps_ini`: hace referencia al número de frames por segundo obtenidos en la primera ejecución, con una distribución de los actores sobre un núcleo.
`double fps`: se refiere al número de frames por segundo obtenidos en cada una de las simulaciones realizadas a lo largo de la ejecución del programa.
`double gain`: hace referencia a la variable donde se va a almacenar la ganancia de cada una de las ejecuciones realizadas.

Se comparan los fps obtenidos durante la ejecución actual, con los fps obtenidos en la ejecución inicial (con una distribución sobre un núcleo) para obtener la ganancia de la ejecución.

El código que representa la parte en la que se realiza la obtención de los resultados es:

```
…
…
while(fgets(linea, LONG_MAX_LINEA, fich_out)!=NULL);
	strcpy (linea_aux, linea);

ultima_linea = strtok (linea_aux," ");
while ((ultima_linea != NULL))
{
	if(i==0)
	{
		strcpy (images_aux, ultima_linea);
		images = atoi (images_aux);
	}
	else if (i==3)
	{
		strcpy (tiempo_aux, ultima_linea);
		tiempo = atof (tiempo_aux);
	}
	else if (i==5)
	{
		strcpy (FPS_aux, ultima_linea);
		fps = atof (FPS_aux);
	}
	i++;
	ultima_linea = strtok (NULL, " ");
	}
…
…
```

6.2.5. Obtención de la ganancia de bloque, discriminación y redistribución de los mismos

En esta fase del programa está encargada de obtener la ganancia de cada uno de los bloques de actores que componen el decodificador. La variable donde se almacena es `double gain_bloque` perteneciente a cada uno de los bloques con la estructura `TBloques` explicada anteriormente. La función que realiza esta tarea es la siguiente:

```
int CargaComputacional(char BloqueAux[TAM_BLOQUE_FIJO], TBloqueGeneral
*BloqueGeneral);
```

Esta función se encarga de generar un fichero de distribución *.xcf (fich_carga.xcf) sobre dos núcleos agrupando todos los bloques de actores menos uno en el núcleo 0 y el bloque restante en el núcleo 1.

Una vez realizada la distribución se ejecutará el decodificador con este fichero y se obtendrán los resultados de su simulación. De esta forma se obtiene la ganancia del bloque situado en el núcleo 1. Este proceso se repite tantas veces como bloques de actores tenga el decodificador, para obtener la ganancia de cada uno de ellos.

Este proceso se repetirá de forma continua mediante un bucle, para cada uno de los bloques del decodificador para obtener la ganancia de ejecución de cada uno de ellos. Este bucle se ejecuta en la función principal de la forma:

```
for (p=0; p<MAX_BINARIO+2; p++)
{
	fflush(stdin);fflush(stdout);
	ObtenerResultados(Param.path_fich_salida);
	CargaComputacional(BloqueFijo, &BloqueCompleto2);
	n_ejecuciones_f(1);
	if (p<MAX_BINARIO+1)
		Ejecutar_Deco(path_new, Param);
	n_ejecuciones_f(3);
}
```

El código que representa una parte de la función que realiza la distribución de los bloques de actores para su ejecución sería el siguiente:

```
…
…
while (fgets(linea, LONG_MAX_LINEA, fichero_aux) != NULL)
{
	if (strstr(linea, "<Partitioning>")!=NULL)
	{
		for (core=0; core < MAX_CORE; core++)
		{
			fprintf(fichero_aux, "<Partition id=%d>\n", core);
			if (core == 0)
```

```
                        {
                                for (d=0, despl_aux1=0x01, despl_aux2=0x1;
                                d<MAX_BINARIO; d++)
                                {
                                        if (((binario1_aux & despl_aux1) !=
                                        despl_aux1))
                                                fputs(BloqueGeneral-
                                                >bloques[d].contenido, fichero_aux);
                                        despl_aux1 = despl_aux1 << 1;
                                }
                                if ((d == 8) && ((binario2_aux & despl_aux2) !=
                                despl_aux2))
                                        fputs(BloqueGeneral->bloques[d].contenido,
                                        fichero_aux);
                        }
                        if (core == 1)
                        {
                                …
                                …
                        }
                        fputs("\n</Partition>\n", fichero_aux);
                }
        }
}
fputs("</Partitioning>\n</Configuration>", fichero_aux);
…
…
```

Una vez que se han obtenido las ganancias de cada uno de los bloques de actores, se procede a realizar una comparación de dichas ganancias para obtener una ganancia mínima.

Con el valor de la ganancia mínima obtenida, se establecerá un umbral de referencia (ganancia min + %) en tanto por ciento sobre dicha ganancia y se establecerá que ganancia no supera este umbral. Esto quiere decir que, si la ganancia de bloque es menor o igual que la ganancia umbral de referencia, se creará un bloque auxiliar con el conjunto de bloques de actores que no superen este umbral.

Por ejemplo: si la ganancia más pequeña obtenida es 0,9 y se establece como porcentaje de aumento un 15%, el límite sería de 0,9 + 15% = 1,035. Todo bloque cuya ganancia no supere este umbral será movido al Bloque Agrupado.

Nota: el tanto por ciento que es sumado a la ganancia mínima puede ser establecido por el usuario, modificando el valor de una variable del tipo #define, que se encuentra en el código C del programa, exactamente en el fichero funciones.c

De esta forma discriminaremos los bloques que pasarán a pertenecer a este bloque auxiliar, denominado Bloque Agrupado.

El objetivo de esta agrupación es reducir el número de bloques de actores que habrá que distribuir entre los núcleos del sistema, reduciendo tanto número de simulaciones a realizar como el tiempo final de ejecución del programa.

La función que realiza esta tarea es:

```
void DiscriminacionBloques(char BloqueAux2[TAM_BLOQUE_AGRUPADO],
TBloqueGeneral *BloqueGeneral)
{
	int m=0;
	double gain_aux_ant=0, gain_aux=0, gain_min=0;

	gain_min = BloqueGeneral->bloques[0].gain_bloque;
	for (m=0; m<bloques; m++)
	{
		gain_aux_ant = gain_aux;
		gain_aux = BloqueGeneral->bloques[m].gain_bloque;
		if ((gain_aux < gain_aux_ant) && gain_aux < gain_min)
			gain_min = gain_aux;
	}
	for (m=0; m<bloques; m++)
	{
		if (BloqueGeneral->bloques[m].gain_bloque <=
		(gain_min+(REFERENCE_GAIN*gain_min)))
		{
			BloqueGeneral->bloques[m].flag_bloqueaux = 1;
			strcat (BloqueAux2, BloqueGeneral->bloques[m].contenido);
			bloques_menor_peso++;
		}
	}
	printf("Bloque Agrupado:\n %s", BloqueAux2);
	for (m=0;m<bloques;m++)
		printf("Bloque %d es: \n%s\n", m, BloqueGeneral-
		>bloques[m].contenido);
}
```

Con la discriminación de bloques realizada, se realizará una redistribución de los mismos.

La redistribución consiste en asignar al array de bloques definido por `TBloqueGeneral *BloqueGeneral` el conjunto de Bloques [n] que no han sido discriminados en la función anterior y finalmente el Bloque Agrupado (con los bloques discriminados).

Esta tarea es llevada a cabo por la siguiente función:

```
void ReorganizarBloques (char BloqueAux2[TAM_BLOQUE_AGRUPADO], TBloqueGeneral
*BloqueGeneral)
{
	int j=0, bloques_en_aux=0;
	for (j=0; j<bloques; j++)
	{
		if (BloqueGeneral->bloques[j].flag_bloqueaux != 1)
```

```
                    strcpy (BloqueGeneral->bloques[j-
                    bloques_en_aux].contenido, BloqueGeneral-
                    >bloques[j].contenido);
              else
                    bloques_en_aux++;
       }
       strcpy (BloqueGeneral->bloques[bloques-bloques_en_aux].contenido,
       BloqueAux2);
       new_tam_bloques = (bloques)-bloques_en_aux+1;
       MAX_BINARIO_AGRU = new_tam_bloques;

       for (j=0; j<new_tam_bloques; j++)
             printf("\nLa reorganizacion del bloque %d es: \n%s\n", j,
             BloqueGeneral->bloques[j].contenido);
}
```

Hay que tener en cuenta, que las tareas que se describen en este apartado, sólo se llevarán a cabo si el usuario selecciona en la ejecución del programa una simulación agrupada.

6.2.6. Crear Máscaras e incrementar número binario

Para continuar con el desarrollo del programa, hay que detenerse a analizar las siguientes dos funciones:

```
void CrearMascaras()
{
       if (new_tam_bloques == 8)
             new_max = 0xFF;
       else if (new_tam_bloques == 7)
             new_max = 0x7F;
       else if (new_tam_bloques == 6)
             new_max = 0x3F;
       else if (new_tam_bloques == 5)
             new_max = 0x1F;
       else if (new_tam_bloques == 4)
             new_max = 0x0F;
       else if (new_tam_bloques == 3)
             new_max = 0x07;
       else if (new_tam_bloques == 2)
             new_max = 0x03;
}
```

Esta función establece el número máximo de ejecuciones según el número total de bloques obtenidos en la primera simulación realizada.

Por ejemplo: si el número de bloques es 8, el número de ejecuciones será de 2^8 por lo que la máscara que se creará para establecer este número máximo de ejecuciones será 0xFF.

La siguiente función incrementa el número binario según el número de ejecuciones llevadas a cabo y la fase del programa en la que se encuentre; esto permitirá realizar la asignación de las instancias a los núcleos, que se detallará en el siguiente apartado.

```
if(!flag_simulacion)
{
	if (binario1 != 0xFF)
		binario1++;
	else
	{
		binario1 = 0x00;
		binario2++;
	}
	if ((binario1 == 0xFF) && (binario2 == 0x01))
	{
		binario1 = 0x00;
		binario2 = 0x00;
		binario3++;
		if (binario3 == 0xFF)
		{
			binario3 = 0x00;
			binario4++;
		}
	}
}
else
{
	CrearMascaras();
	if (binario1 != new_max)
		binario1++;
	else
	{
		binario1 = 0x00;
		binario2++;
	}
}
…
…
```

6.2.7. Asignación de núcleos

El proceso de asignación de los bloques de actores en los núcleos es llevado a cabo por la siguiente función:

```
int AsignarNucleos (TBloqueGeneral *BloqueGeneral);
```

Consiste en que, según el número de ejecución en que se encuentre el programa y por lo tanto según el valor binario de ejecución que haya en ese momento, se realiza la asignación del núcleo donde posteriormente se escribirá el bloque. Esto se explicará más detalladamente a lo largo de este sub-apartado.

Para explicar esto de forma más detallada y precisa, se realizará el siguiente ejemplo trabajando con el decodificador Mpeg_part10_CBP y sus actores con una distribución de dos núcleos.

El diagrama de bloques del nivel superior de este codificador sería el de la Fig.39

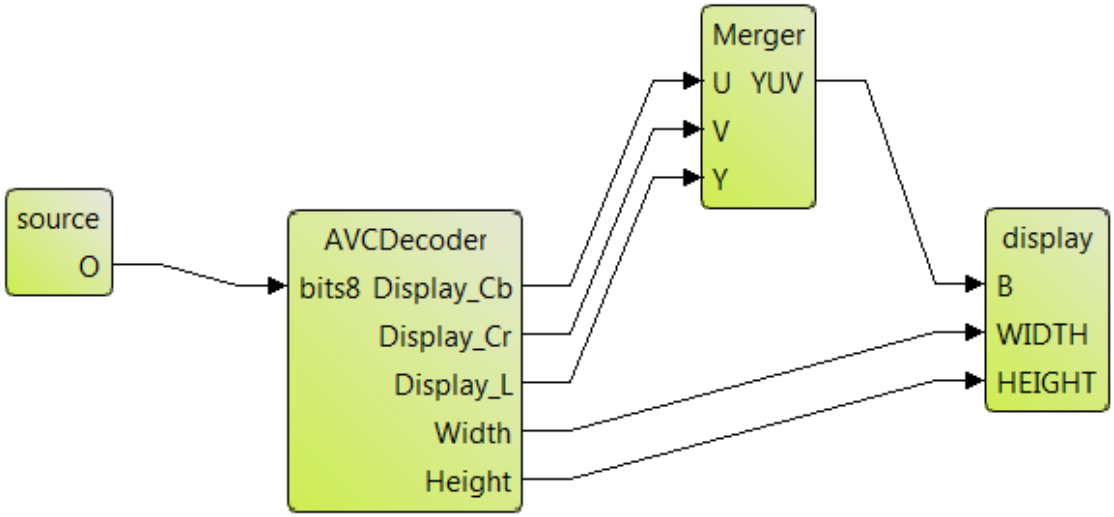


Figura 40. Diagrama de bloques Top_MPEG4_part10_CBP_decoder

En la Fig.40 se pueden ver las conexiones (en su nivel más alto) entre los diferentes bloques que posee el decodificador. Internamente, cada bloque se divide en sub-bloques y cada uno de ellos posee una descripción en lenguaje CAL.

Este decodificador posee un total de 9 Bloques de actores que son los siguientes:

- Bloque Parser (P), que contiene 10 actores.
- Bloque Luminancia (Y), que contiene 33 actores.
- Bloque Crominancia (U), que contiene 21 actores.
- Bloque Crominancia (V), que contiene 21 actores.
- Bloque Display (D), que contiene 4 actores.
- Bloque Merger (M), que contiene 1 actor.
- Bloque Source (S), que contiene 1 actor.
- Bloque Cavlc (C), que contiene 2 actores.
- Bloque display (d), que contiene 1 actor.

La primera vez que se ejecuta la función que realiza la obtención de bloques se obtienen 9 bloques y la asignación en referencia con el número binario sería la siguiente:

	Binario 2								**Binario 1**							
Nº binario	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_	_
Distribución de actores	_	_	_	_	_	_	_	d	C	S	M	D	V	U	Y	P

Aclarar, que las partes en las que aparece el símbolo _ son partes del binario que no son utilizadas debido a que en este caso no hay suficientes actores para completar Binario 2.

En un ejemplo concreto, la asignación de núcleos sería la siguiente:
Si se tienen el número binario: Binario 2 = 0x01 y Binario 1 = 0xA3, se obtendría:

	Binario 2								**Binario 1**							
Nº binario	0	0	0	0	0	0	0	1	1	0	1	0	0	0	1	1
Distribución de actores	_	_	_	_	_	_	_	d	C	S	M	D	V	U	Y	P

En el núcleo 0 se escribirían los siguientes bloques:

- Crominancia U.
- Crominancia V.
- Display.
- Source.

En el núcleo 1 se escribirían los siguientes bloques:

- Parser.
- Luminancia Y.
- Merger.
- Cavlc
- display

Esta asignación de los actores depende principalmente del número de actores de cada decodificador así como del número de núcleos sobre los que se quiera realizar la simulación. Esto quiere decir que el número de Binarios a utilizar se incrementaría en relación a estos parámetros.

La asignación del núcleo en el que se escribirá cada bloque se realiza de la siguiente forma:

```
BloqueGeneral->bloques[k].nucleo = 1;
```

Todo este proceso queda definido en el siguiente código:

```
…
…
if (MAX_CORE < 3)
{
	for (k=0, despl1=0x01, despl2=0x01; k < MAX_BINARIO; k++)
	{
		if ((binario1 & despl1) != despl1)
			BloqueGeneral->bloques[k].nucleo = 0;
		else
			BloqueGeneral->bloques[k].nucleo = 1;
		despl1 = despl1 << 1;
	}

	if (k == 8)
```

```
	{
		if ((binario2 & despl2) != despl2)
			BloqueGeneral->bloques[k].nucleo = 0;
		else
			BloqueGeneral->bloques[k].nucleo = 1;
	}
}
else if (MAX_CORE == 3)
{
	for (k=0, despl1=0x01, despl2=0x01, despl3=0x01, despl4=0x01; k <
	MAX_BINARIO; k++)
	{
		if (((binario1 & despl1) != despl1) && ((binario3 & despl3) !=
		despl3))
			BloqueGeneral->bloques[k].nucleo = 0;
		else if (((binario1 & despl1) == despl1) && ((binario3 & despl3)
		!= despl3))
			BloqueGeneral->bloques[k].nucleo = 1;
		else if (((binario1 & despl1) != despl1) && ((binario3 & despl3)
		== despl3))
			BloqueGeneral->bloques[k].nucleo = 2;
		else if (((binario1 & despl1) == despl1) && ((binario3 & despl3)
		== despl3))
			saltar_ejecucion = 1;
		despl1 = despl1 << 1; despl3 = despl3 << 1;
	}
	if (k == 8)
	{
		if (((binario2 & despl2) != despl2) && ((binario4 & despl4) !=
		despl4))
			BloqueGeneral->bloques[k].nucleo = 0;
		else if (((binario2 & despl2) == despl2) && ((binario4 & despl4)
		!= despl4))
			BloqueGeneral->bloques[k].nucleo = 1;
		else if (((binario2 & despl2) != despl2) && ((binario4 & despl4)
		== despl4))
			BloqueGeneral->bloques[k].nucleo = 2;
		else if (((binario2 & despl2) == despl2) && ((binario4 & despl4)
		== despl4))
			saltar_ejecucion = 1;
	}
}
else if (MAX_CORE > 3)
{
...
...
```

6.2.8. Generar nuevo fichero de distribución

Esta fase del programa se encarga de crear el fichero *.xcf con la nueva distribución de los bloques una vez realizada la asignación de cada uno de los bloques en los respectivos núcleos.

Primero se genera el nuevo fichero de distribución *.xcf, a continuación se abre el fichero y se escribe el Bloque fijo (que se obtuvo anteriormente). Después se comienza a analizar la variable `nucleo` de cada uno de los bloques escribiendo el contenido del bloque en el núcleo indicado por esta variable. Cuando este proceso haya finalizado se obtendrá el nuevo fichero de distribución *.xcf

Esto se observa en la siguiente parte de código de la función:

…
…

```
if(!flag_simulacion)
{
	while (fgets(linea, LONG_MAX_LINEA, fich) != NULL)
	{
		if (strstr(linea,"<Partitioning>")!=NULL)
		{
			for (core=0; core < MAX_CORE; core++)
			{
				fprintf(fich, "<Partition id=%d>\n", core);
				if (core == n)
				{
					for (d=0; d<MAX_BINARIO+1;d++)
					{
						if (BloqueGeneral->bloques[d].nucleo
						== 0)
							fputs(BloqueGeneral-
							>bloques[d].contenido, fich);
					}
				}
				fputs("\n</Partition>\n", fich);
			}
		}
	}
	fputs("</Partitioning>\n</Configuration>", fich);
}
```

…
…

6.2.9. Guardar configuración de distribución

En esta fase del programa, se establecerá un valor de ganancia mínimo (ganancia umbral), con la que se discriminarán las distribuciones que consiguen un buen rendimiento, de las que no.

Esta ganancia umbral está definida en el programa de la forma: `#define gain`, y puede ser modificada por el usuario en cualquier momento, cambiando su valor. Se ha establecido un valor de ganancia umbral mayor o igual a 1,8 en simulaciones sobre 2 núcleos y una ganancia umbral de 1,9 en simulaciones de 3 o más núcleos. Este límite de ganancia dependerá tanto del descodificador como de la arquitectura que posea, por lo que será necesario modificarlo, ajustarlo según el descodificador utilizado en la ejecución. Si al realizar la simulación con cada nueva distribución de los actores sobre los núcleos, los datos obtenidos son menores que este límite, el programa indicará que la distribución no es adecuada y volverá a realizar una nueva distribución de los actores. Si por el contrario se obtiene una ganancia igual o mayor a la ganancia umbral establecida, se creará un nuevo fichero de texto en el que se irán almacenando las nuevas configuraciones de distribución así como los datos obtenido de su rendimiento.

Una vez finalizada la ejecución de todas las simulaciones posibles, se habrá generado un fichero de texto con las mejores distribuciones de los actores en los núcleos obtenidas con los valores de número de imágenes, tiempo transcurrido, frames por segundo y la ganancia.

En este fichero de texto, cada resultado estará estructurado de la siguiente forma:

- Distribución de los actores en los diferentes núcleos.
- Número de imágenes obtenidas.
- Tiempo transcurrido en la ejecución.
- Número de *frames* por segundo.
- Resultado de la ganancia obtenida para esa distribución.

Cada nuevo resultado de una buena distribución de los actores irá a continuación uno de otro.

A continuación, la Fig.40 mostrará un ejemplo de uno de los resultados del fichero de texto generado por el programa.

```
    <?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Partitioning>
<Partition id=0>
            <Instance id="AVCDecoder_Decode_V_ResidualC_Algo_Merge_4x4_to_8x8"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_Algo_Merge_4x4_to_8x8"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_Algo_IS_Zigzag_4x4"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_Mgnt_IQ_Chroma"/>
            <Instance id="AVCDecoder_Decode_V_Buffer_inter"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Select_chroma"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Deblocking_Filter"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_DemuxParserInfos"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Intra8x8_Algo_IntraPred_CHROMA"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Intra8x8_Algo_Add"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Intra8x8_Buffer_Neighbour_FullMb"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Intra8x8_Mgnt_Intra_8x8"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Inter_Algo_Interp_EighthPelBilinear"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Inter_Algo_Add"/>
            <Instance id="AVCDecoder_Decode_V_PredictionC_Inter_Algo_Interp_Reord"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_Algo_DCR_Hadamard_CHROMA"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_Algo_IQ_QSAndSLAndIDCTScaler_4x4"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_0"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_1"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decode_V_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_Addshift"/>
           <Instance id="AVCDecoder_Display_Render_DisplayRendererV"/>
           <Instance id="AVCDecoder_Display_Render_Mgnt_Display_Poc"/>
           <Instance id="AVCDecoder_Display_Render_DisplayRendererY"/>
           <Instance id="AVCDecoder_Display_Render_DisplayRendererU"/>
           <Instance id="AVCDecoder_CavlcExpand_BlockExpand"/>
           <Instance id="AVCDecoder_CavlcExpand_BlockSplit"/>
           <Instance id="display"/>

</Partition>
<Partition id=1>
           <Instance id="AVCDecoder_Syn_Parser_Algo_Synp"/>
           <Instance id="AVCDecoder_Syn_Parser_Algo_Parser_IPCM"/>
           <Instance id="AVCDecoder_Syn_Parser_IntraPredSplit"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MMCO"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_RefList0"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_Mgnt_InterPred"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MgntDeblockingFilter"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvL0_Reconstr"/>
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_RefIdxL0ToFrameNum"/:
           <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvComponentReord"/>
           <Instance id="AVCDecoder_Decode_U_Buffer_inter"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Select_chroma"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Deblocking_Filter"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_DemuxParserInfos"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Algo_IntraPred_CHROMA"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Algo_Add"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Buffer_Neighbour_FullMb"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Mgnt_Intra_8x8"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Interp_EighthPelBilinear"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Add"/>
           <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Interp_Reord"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_Algo_DCR_Hadamard_CHROMA"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_Algo_Merge_4x4_to_8x8"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Algo_IS_Zigzag_4x4"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Mgnt_IQ_Chroma"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Algo_IQ_QSAndSLAndIDCTScaler_4x4"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_0"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_0"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_1"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_1"/>
           <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_Addshift"/>
            <Instance id="Merger"/>
            <Instance id="source"/>

</Partition>
<Partition id=2>
```

```
            <Instance id="AVCDecoder_Decoding_Y_DecodedPictureBuffer"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Select"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Deblocking_Filter"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_DemuxParserInfos"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Mgnt_Intra_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Algo_IntraPred_LUMA_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Add_Clip"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Buffer_Neighbour_FullMb"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Mgnt_Intra4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_IntraPred_LUMA_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Add_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Merge_4x4_to_16x16_norasterscan"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Split_16x16_to_4x4_norasterscan"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Buffer_Neighbour_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Algo_Interp_SeparableSixTapQuarterPelAVC"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Add_Clip"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Algo_Interp_Reord"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_Algo_Merge_4x4_to_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_IS_Zigzag_4x4_DC"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_IHT1d_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DC_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_IHT1d_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_Scaling"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DC_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_Reordering"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Algo_IS_Zigzag_4x4_AC"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Mgnt_IQ_INTRA16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Algo_IQ_QSAndSLAndIDCTScaler_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_1d_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_1d_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_Addshift"/>
</Partition>
</Partitioning>
</Configuration>

////////////////////////////////////
The number of images is: 234
The elapsed time is: 2.357
The number of FPS is: 99.2787
The gain achieved is: 2.52117
////////////////////////////////////
```

Figura 41. Ejemplo del resultado del fichero.txt generado por el programa

La función que implementa esta funcionalidad es la siguiente:

```
if (((MAX_CORE < 3) && (gain >= 1.85)) || ((MAX_CORE >= 3) && (gain >= 1.9)))
{
	if ((fich_escri = fopen(fich1, "a+")) == NULL)
	{
		perror("Resultados.txt");
		return EXIT_FAILURE;
	}
	if ((fich_lect = fopen(fich2, "r")) == NULL)
	{
		perror("New_fich_xcf.xcf");
		return EXIT_FAILURE;
	}

	fseek(fich_escri, 0, SEEK_END);
	fprintf(fich_escri, "//////////////////Nueva
	Distribucion//////////////////\n\n");

	while (fgets(linea, LONG_MAX_LINEA, fich_lect) != NULL)
		fputs(linea, fich_escri);
```

```
fprintf(fich_escri, "\n\n////////////////////////////////////\n");
fprintf(fich_escri, "The number of images is: %d\n", images);
fprintf(fich_escri, "The elapsed time is: %g\n", tiempo);
fprintf(fich_escri, "The number of FPS is: %g\n", fps);
fprintf(fich_escri, "The gain achieved is: %g\n", gain);
fprintf(fich_escri, "////////////////////////////////////\n\n");
```

CAPÍTULO 7. BANCO DE TEST

En este apartado se van a exponer y analizar los resultados de las pruebas obtenidos a lo largo del estudio realizado sobre los decodificadores MPEG 4 part2, MPEG 4 parte 10 CBP y PHP, y HEVC, así como los obtenidos una vez finalizado el desarrollo del programa. Se estudiarán los mejores resultados obtenidos y se explicará el motivo por el que se obtienen dichos resultados. En este capítulo se realizará una comparación entre los resultados de ganancia obtenidos realizando la distribución manual por parte de un experto y los obtenidos mediante el programa desarrollado.

El capítulo está dividido en diferentes apartados; en el primero se realiza una comparativa de los resultados obtenidos de forma manual y utilizando el programa para cada uno de los decodificadores mencionados anteriormente. El segundo apartado abarca los resultados de los tiempos de ejecución con cada decodificador, distribución y vídeo utilizado; y el último capítulo recoge las conclusiones finales.

7.1 Resultados ejecución

En este apartado se mostrarán el conjunto de resultados obtenidos durante el análisis y estudio de cada decodificador así como cuales son las mejores distribuciones obtenidas tanto de forma manual (siguiendo el criterio del desarrollador, basado en el estudio y análisis tanto de los decodificadores como de sus actores) como las obtenidas por el programa desarrollado. También se analizarán el peso, la importancia de cada grupo de actores sobre varios núcleos.

Se seguirá un proceso de análisis progresivo, es decir, desde el decodificador más básico hasta el más complejo que se ha estudiado.

Nota: como ya se explicó anteriormente, el cálculo de la ganancia se ha realizado de la siguiente forma: *gain = fps / fps_ini*

Indicar que todas las pruebas realizadas se han llevado a cabo con el mismo PC, el cual posee estas características principales:

- Procesador Intel Core i7 CPU 950 @ 3,07Ghz x 8
- Disco duro de 500GB
- Memoria RAM de 6GB
- Linux Ubuntu 12.04 LTS de 64-bit

7.1.1 Resultados decodificador MPEG4 parte 2 SP

Se ha utilizado el decodificador MPEG4 parte 2 SP con tres vídeos de muestra (*foreman_cif_xvid_384kbps_I_P.m4v*, *chapeau_melon_PAL_IP_track1.avi* y *5_element_xvid_80_64_512kbps_intra.m4v*) y con distribuciones de 1, 2 y 3 núcleos. El número de actores que posee este decodificador es de 41 y los bloques obtenidos en la ejecución del programa es de 9. En este apartado se mostrarán los mejores resultados obtenidos realizando las diferentes distribuciones de los actores en los núcleos de forma manual y se compararán con los resultados obtenidos mediante el programa. Todos los resultados obtenidos se podrán ver y analizar más detalladamente en el resumen del Anexo 3.

Nota: *Las pruebas con 4 o más núcleos no se han podido llevar a cabo (para ninguno de los decodificadores utilizados) debido a que al haber un número de combinaciones tan elevado, el tiempo de ejecución era altísimo, alrededor de 2 a 3 semanas, dependiendo del decodificador (ya que varía el número de actores) y vídeo escogidos.*

i. Resultados para vídeo *foreman_cif_xvid_384kbps_I_P.m4v:*

El vídeo tiene una resolución de 352x288 y decodifica 298 imágenes en cada ejecución. El vídeo se muestra en la Figura 42.

Figura 42. Secuencia vídeo *foreman_cif_xvid_384kbps_I_P.m4v*

Nota: *todas las tablas que aparecen a lo largo de este capítulo, muestran los resultados de ejecución del vídeo correspondiente, la distribución y el número de núcleos sobre el que se ha llevado a cabo dicha ejecución, tiempo de ejecución, frames por segundo obtenidos, así como la ganancia obtenida. El número de veces que se repite cada ejecución es de 1.*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	2,864	104,708	1,000
2	N0: textureY+motionY+textureU N1: resto	1,107	269,196	2,571
2	N0: textureY+motionY+textureV N1: resto	1,096	271,898	2,597
2	N0: textureY+motionY N1: resto	1,279	232,995	2,225
3	N0:textureY+motionY N1: texture y motionU+V N2: resto	1,046	284,895	2,721
3	N0: textureU N1:textureV N2: resto	1,837	162,221	1,549
3	N0: Parser+textureU N1: merger+textureV N2: resto	1,229	242,474	2,316

Tabla 1. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	2,846	104,708	1,000
2	N0: textureY+motionY+textureU N1: resto	1,107	269,196	2,571
2	N0: textureY+motionY+textureV N1: resto	1,096	271,898	2,597
3	N0: Parser N1: textureY+motionY N2: resto	0,893	333,707	3,187
3	N0: Parser+Merger N1: textureY+motionY N2: resto	1,028	289,882	2,768
3	N0: Parser+Merger+textureU N1: textureY+motionY N2: resto	1,072	277,985	2,655

Tabla 2. Mejores resultados obtenidos mediante programa (simulación completa)

ii. Resultados para vídeo *chapeau_melon_PAL_IP_track1.avi:*

El vídeo tiene una resolución de 720 x 576 y decodifica 249 imágenes en cada ejecución. El vídeo se muestra en la Figura 43.

Figura 43. Secuencia de vídeo *chapeau_melon_PAL_IP_track1.avi*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	6,490	38,367	1,000
2	N0: textureY+motionY+textureU N1: resto	3,861	64,491	1,681
2	N0: textureY+motionY+textureV N1: resto	3,823	65,132	1,698
2	N0: Parser N1: textureY+motionY N2: resto	3,846	64,743	1,687
3	N0: textureY+motionY N1: textureU+V N2: resto	3,175	78,425	2,044
3	N0: textureU N1: textureV N2: resto	5,948	41,863	1,091
3	N0: Parser+textureU N1: merger+textureV N2: resto	3,799	65,544	1,708

Tabla 3. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1núcleo)	6,490	38,367	1,000
2	N0: textureY+motionY+textureU N1: resto	3,861	64,491	1,681
2	N0: textureY+motionY+textureV N1: resto	3,823	65,132	1,698
2	N0: textureY+motionY N1: resto	3,741	66,560	1,735
3	N0: Parser N1: textureY+motionY N2: resto	3,431	72,574	1,892
3	N0: Parser+merger N1: textureY+motionY N2: resto	3,211	77,546	2,021
3	N0: Parser+Merger+U N1:textureY+motionY N2: resto	3,205	77,691	2,025

Tabla 4. Mejores resultados obtenidos mediante programa (simulación completa)

iii. Resultados para vídeo *5_element_xvid_80_64_512kbps_intra.m4v*:

El vídeo tiene una resolución de 80 x 64 y decodifica 2695 imágenes en cada ejecución. El vídeo se muestra en la Figura 44

Figura 44. Secuencia de vídeo *5_element_xvid_80_64_512kbps_intra.m4v*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	5,317	506,865	1,000
2	N0: textureY+motionY+textureU N1: resto	2,013	1338,798	2,641
2	N0: textureY+motionY+textureV N1: resto	2,050	1314,634	2,594
2	N0: textureY+motionY N1: resto	2,788	966,643	1,907
3	N0: textureY+motionY N1: textureU+V N2: resto	1,868	1442,719	2,846
3	N0: textureU N1: textureV N2: resto	2,465	1093,306	2,157
3	N0: Parser+textureU N1: merger+textureV N2: resto	2,125	1286,235	2,538

Tabla 5. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1núcleo)	5,317	506,865	1,000
2	N0: textureY+motionY+textureU N1: resto	2,013	1338,798	2,641
2	N0: textureY+motionY+textureV N1: resto	2,050	1314,634	2,594
2	N0: texture-motionY+texture-motionU y Y N1: resto	2,079	1296,296	2,557
3	N0: Parser N1: textureY+motionY N2: resto	1,792	1503,906	2,967
3	N0: Parser+merger N1: textureY+motionY N2: resto	1,676	1607,995	3,172
3	N0: Parser+Merger+U N1:textureY+motionY N2: resto	1,640	1643,293	3,242

Tabla 6. Mejores resultados obtenidos mediante programa (simulación completa)

7.1.2 Resultados decodificador MPEG4 parte 10 CBP

Se ha utilizado el decodificador MPEG4 parte 10 CBP con tres vídeos de muestra (*LS_SVA_D.264*, *HCBP1_HHI_A.264* y *HCBP2_HHI_A.264*) y con distribuciones de 1, 2 y 3 núcleos. El número de actores que posee este decodificador es de 94 y los bloques obtenidos en la ejecución del programa es de 9. En este apartado se mostrarán los mejores resultados obtenidos realizando las diferentes distribuciones de los actores en los núcleos de forma manual y se compararán con los resultados obtenidos mediante el programa. Todos los resultados obtenidos se podrán ver y analizar más detalladamente en el resumen del Anexo 3.

i. Resultados para vídeo *LS_SVA_D.264*:

El vídeo tiene una resolución de 176 x 144 y decodifica 1687 imágenes en cada ejecución. El vídeo se muestra en la Figura 45.

Figura 45. Secuencia de vídeo *LS_VSA_D.264*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	11,040	152,808	1,000
2	N0: Parser+merger+Y N1: resto	4,617	365,172	2,390
2	N0: U+V N1: resto	5,396	312,639	2,046
2	N0: U+V+Y N1: resto	5,388	313,103	2,049
3	N0: Parser N1: Y N2: resto	3,343	504,637	3,302
3	N0: Y+U N1: Parser+V N2: resto	4,329	389,697	2,550
3	N0: Parser+Y N1: U+V N2: resto	4,417	381,933	2,499

Tabla 7. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	11,040	152,808	1,000
2	N0: Parser+Y N1: resto	4,006	421,118	2,756
2	N0: Y,U N1: resto	4,105	410,962	2,689
2	N0: Y,V N1: resto	4,162	405,334	2,653
3	N0: Parser+Merger N1: Y N2: resto	3,290	512,766	3,356
3	N0: Parser+Merger+U N1: Y N2: resto	3,126	539,667	3,532
3	N0: Parser+Y+merger+source+cavlc N1: U+V N2: resto	3,290	512,766	3,356

Tabla 8. Mejores resultados obtenidos mediante programa (simulación completa)

ii. Resultados para vídeo *HCBP1_HHI_A.264*:

El vídeo tiene una resolución de 352 x 288 y decodifica 234 imágenes en cada ejecución. El vídeo se muestra en la Figura 46.

Figura 46. Secuencia de vídeo *HCBP1_HHI_A.264*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	6,290	37,043	1,000
2	N0: Parser+merger+Y N1: resto	3,811	61,139	1,650
2	N0: U+V N1: resto	4,203	55,675	1,503
2	N0: U+V+Y N1: resto	4,196	55,529	1,499
3	N0: Parser N1: Y N2: resto	2,453	95,393	2,575
3	N0: Y+U N1: Parser+V N2: resto	2,450	95,510	2,578
3	N0: Parser+Y N1: U+V N2: resto	3,995	58,573	1,581

Tabla 9. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	6,290	37,043	1,000
2	N0: Y N1: resto	2,932	79,468	2,145
2	N0: Y+U N1: resto	3,111	75,217	2,031
2	N0: Parser+U+V N1: resto	2,886	80,735	2,179
3	N0: Parser N1: Y N2: resto	2,462	95,045	2,566
3	N0: Parser+Merger N1: Y N2: resto	2,444	95,745	2,585
3	N1: Parser+Merger+U N1: Y N2: resto	2,387	98,031	2,646

Tabla 10. Mejores resultados obtenidos mediante programa (simulación completa)

iii. Resultados para vídeo *HCBP2_HHI_A.264*:

El vídeo tiene una resolución de 352 x 288 y decodifica 232 imágenes en cada ejecución. El vídeo se muestra en la Figura 47

Figura 47. Secuencia de vídeo *HCBP2_HHI_A.264*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	4,802	48,313	1,000
2	N0: Parser+merger+Y N1: resto	2,701	85,894	1,778
2	N0: U+V N1: resto	2,988	77,644	1,607
2	N0: U+V+Y N1: resto	3,356	69,428	1,437
3	N0: Parser N1: Y N2: resto	2,012	115,308	2,387
3	N0: Y+U N1: Parser+V N2: resto	2,012	115,308	2,387
3	N0: Parser+Y N1: U+V N2: resto	2,667	87,364	1,808

Tabla 11. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	4,802	48,313	1,000
2	N0: Y N1: resto	2,424	96,122	1,990
2	N0: Y+U N1: resto	2,527	92,204	1,908
2	N0: Parser+U+V N1: resto	2,367	98,014	2,029
3	N0: Parser N1: Y N2: resto	2,021	114,795	2,376
3	N0: Parser+Merger N1: Y N2: resto	2,006	115,653	2,394
3	N1: Parser+Merger+U N1: Y N2: resto	1,937	120,289	2,490

Tabla 12. Mejores resultados obtenidos mediante programa (simulación completa)

7.1.3 Resultados decodificador MPEG4 parte 10 PHP

En este caso se ha utilizado el decodificador MPEG4 parte 10 PHP con tres vídeos de muestra (*LS_SVA_D.264*, *HCBP1_HHI_A.264* y *HCBP2_HHI_A.264*) y con distribuciones de 1, 2 y 3 núcleos. El número de actores que posee este decodificador es de 114 y los bloques obtenidos en la ejecución del programa es de 9. En este apartado se mostrarán los mejores resultados obtenidos realizando las diferentes distribuciones de los actores en los núcleos de forma manual y se compararán con los resultados obtenidos mediante el programa. Todos los resultados obtenidos se podrán ver y analizar más detalladamente en el resumen del Anexo 3.

Nota: *Los vídeos utilizados para la realización de las pruebas de este apartado son los mismos que en el apartado anterior.*

i. Resultados para vídeo *LS_SVA_D.264*:

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	14,643	115,209	1,000
2	N0: Parser+merger+Y N1: resto	11,114	151,701	1,317
2	N0: U+V N1: resto	6,624	254,680	2,211
2	N0: U+V+Y N1: resto	6,322	266,846	2,316
3	N0: Parser+source+merger+cavlc N1: Y N2: resto	8,741	192,999	1,675
3	N0: Y+U N1: Parser+V N2: resto	9,543	176,779	1,534
3	N0: Y N1: Parser+U N2: resto	8,697	193,975	1,684

Tabla 13. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	14,643	115,209	1,000
2	N0: Y N1: resto	4,849	347,907	3,020
2	N0: Y+U N1: resto	4,794	351,898	3,054
2	N0: Parser+U+V N1: resto	4,583	368,099	3,195
3	N0: Parser N1: Y N2: resto	3,752	449,627	3,903
3	N0: Parser+Merger N1: Y N2: resto	3,704	445,454	3,866
3	N1: Parser+Merger+U N1: Y N2: resto	3,638	463,716	4,025

Tabla 14. Mejores resultados obtenidos mediante programa (simulación completa)

ii. Resultados para vídeo *HCBP1_HHI_A.264*:

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	8,178	28,491	1,000
2	N0: Y+U N1: resto	3,719	62,651	2,199
2	N0: Y N1: resto	3,713	62,752	2,203
2	N0: U+V+Y N1: resto	4,899	47,765	1,676
3	N0: Parser+display N1: Y N2: resto	3,433	67,871	2,382
3	N0: Parser+U+display N1: Y N2: resto	3,583	65,029	2,282
3	N0: Parser+U N1: V+display N2: resto	7,233	32,352	1,136

Tabla 15. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	8,178	28,491	1,000
2	N0: Y N1: resto	3,713	62,752	2,203
2	N0: Y+V N1: resto	3,697	63,295	2,222
2	N0: Parser+U+V N1: resto	3,257	71,538	2,511
3	N0: Parser N1: Y N2: resto	2,837	82,481	2,895
3	N0: Parser+Merger N1: Y N2: resto	2,895	80,829	2,837
3	N1: Parser+Merger+U N1: Y N2: resto	2,778	84,233	2,956

Tabla 16. Mejores resultados obtenidos mediante programa (simulación completa)

iii. Resultados para vídeo *HCBP2_HHI_A.264*:

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	10,550	21,896	1,000
2	N0: Y+U N1: resto	5,231	44,160	2,017
2	N0: Y N1: resto	4,474	51,632	2,358
2	N0: U+V+Y N1: resto	7,135	32,376	1,479
3	N0: Y+U N1: Parser+V N2: resto	4,363	52,945	2,418
3	N0: Y N1: Parser+U N2: resto	4,938	46,780	2,136

Tabla 17. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(P)
1	Predeterminado (1 núcleo)	10,550	21,896	1,000
2	N0: Y N1: resto	4,592	50,305	2,297
2	N0: Y+V N1: resto	5,241	44,076	2,013
2	N0: Parser+U+V N1: resto	4,181	55,250	2,523
3	N0: Parser N1: Y N2: resto	3,869	59,705	2,727
3	N0: Parser+Merger N1: Y N2: resto	3,890	59,383	2,712
3	N1: Parser+Merger+U N1: Y N2: resto	3,762	61,404	2,804

Tabla 18. Mejores resultados obtenidos mediante programa (simulación completa)

7.1.4 Conclusiones sobre los resultados del decodificador MPEG

A la vista de los resultados obtenidos, se puede asegurar que para los decodificadores MPEG4 parte 2 y parte 10 CBP y PHP, los bloques de actores que más peso tienen son la luminancia (Y) y las crominancias U y V. Estos bloques de actores son paralelizables, lo que facilita la tarea de distribución en los núcleos. Esto se ha podido comprobar con los mejores resultados obtenidos para cada decodificador y con los resultados de las mejores distribuciones posibles.

Para analizar tanto la funcionalidad como la eficacia del programa desarrollado, se va a realizar una comparativa de los mejores resultados de ganancia obtenidos de forma manual frente a los obtenidos mediante el programa, para los diferentes núcleos con la mejor de las distribuciones obtenidas mediante los decodificadores Mpeg 4. Esto se muestra en la siguiente Tabla 19.

Decoder	Secuencia	Core	Gain(M)	Gain(P)	Mejora
Mpeg part2 SP	foreman_cif_xvid_384kbps_I_P.m4v	2	2,574	2,597	0,89%
Mpeg part2 SP	foreman_cif_xvid_384kbps_I_P.m4v	3	2,721	3,187	17,13%
Mpeg part2 SP	chapeau_melon_PAL_IP_track1.avi	2	1,698	1,992	17,31%
Mpeg part2 SP	chapeau_melon_PAL_IP_track1.avi	3	2,044	2,871	40,46%
Mpeg part2 SP	5_element_xvid_80_64_512kbps_intra.m4v	2	1,907	2,641	38,49%
Mpeg part2 SP	5_element_xvid_80_64_512kbps_intra.m4v	3	2,538	3,242	27,74%
Mpeg part10 CBP	LS_SVA_D.264	2	2,049	2,756	34,50%
Mpeg part10 CBP	LS_SVA_D.264	3	2,550	3,532	38,51%
Mpeg part10 CBP	HCBP1_HHI_A.264	2	1,650	2,179	32,06%
Mpeg part10 CBP	HCBP1_HHI_A.264	3	2,220	2,585	16,44%
Mpeg part10 CBP	HCBP2_HHI_A.264	2	1,778	2,029	14,12%
Mpeg part10 CBP	HCBP2_HHI_A.264	3	2,387	2,490	4,32%
Mpeg part10 PHP	LS_SVA_D.264	2	2,316	3,195	37,95%
Mpeg part10 PHP	LS_SVA_D.264	3	1,784	4,025	125,62%
Mpeg part10 PHP	HCBP1_HHI_A.264	2	2,203	2,511	13,98%
Mpeg part10 PHP	HCBP1_HHI_A.264	3	2,382	2,956	24,10%
Mpeg part10 PHP	HCBP2_HHI_A.264	2	2,358	2,523	7,00%
Mpeg part10 PHP	HCBP2_HHI_A.264	3	2,418	2,804	15,96%

Tabla 19. Comparación de los resultados de ganancia Decodificador Mpeg 4

Como se puede observar en la tabla anterior, los mejores resultados se obtienen mediantes distribuciones en 3 núcleos frente a 2 núcleos, independientemente del decodificador y vídeo utilizados. Si se hace referencia a todos los resultados obtenidos mediante los decodificadores Mpeg 4, se puede afirmar, que el conjunto de actores que tienen más carga, y por lo tanto más relevancia son los que forman la luminancia Y, seguidos por las dos crominancias U y V. Realizando una buena distribución de estos actores, se puede conseguir aumentar considerablemente la ganancia obtenida frente a los resultados para 1 núcleo.

También, haciendo uso de los resultados de la Tabla 19, se puede ver la eficacia y la importancia del programa desarrollado. Independientemente del decodificador que se utilice y el vídeo a decodificar, mediante el programa se consigue en todos los casos una mejoría con respecto a los resultados obtenidos de forma manual por el usuario. Esto quiere decir que, aunque el usuario posea grandes conocimientos sobre el decodificador y realice buenas distribuciones de sus actores sobre los núcleos, el programa prácticamente siempre obtendrá al menos una distribución mejor y más eficiente, sin la necesidad de la intervención del usuario. Este hecho resalta tanto la eficacia como la comodidad de la aplicación desarrollada.

7.1.5 Resultados codificador HEVC

En este apartado se ha utilizado el decodificador HEVC con cuatro vídeos de muestra (*LD_BasketballDrill_832x480_50_qp27.bin*, *RA_BasketballDrill_832x480_50_qp27.bin, LD_RaceHorses_832x480_30_qp27.bin* y RA_*RaceHorses_832x480_30_qp27.bin*) con distribuciones de 1, 2 y 3 núcleos. El número de actores que posee este decodificador es de 38 y los bloques obtenidos en la ejecución del programa es de 12. En este apartado se mostrarán los mejores resultados obtenidos realizando las diferentes distribuciones de los actores en los núcleos de forma manual y se compararán con los resultados obtenidos mediante el programa. Todos los resultados obtenidos se podrán ver y analizar más detalladamente en el resumen del Anexo 3.

Nota: *Actualmente el decodificador HEVC está en continuo cambio y desarrollo para mejorar su eficiencia. En este proyecto se ha empleado una versión de este decodificador de Enero de 2014.*

i. Resultados para vídeo *LD_BasketballDrill_832x480_50_qp27.bin*

El vídeo tiene una resolución de 832 x 480 y decodifica 496 imágenes en cada ejecución. El vídeo se muestra en la Figura 48.

Figura 48. Secuencia de vídeo *LD_BasketballDrill_832x480_50_qp27.bin*

Cores	***.xcf***	***time(s)***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	18,141	27,341	1,000
2	N0: Inter+xIT N1: resto	12,729	38,966	1,425
2	N0: Intra+xIT N1: resto	16,257	30,510	1,116
2	N0: Inter+xIT+source+display N1: resto	12,446	39,852	1,458
3	N0: Inter+xIT N1: source+display+merge N2: resto	12,525	39,601	1,448
3	N0: Inter N1: Intra N2: resto	14,784	33,550	1,227

Tabla 20. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	18,141	27,341	1,000
2	N0: Inter+qpGen+Sao+xIT N1: resto	11,296	43,909	1,606
2	N0: Inter+Sao+xIT N1: resto	11,322	43,809	1,602
2	N0: Source+Inter+QpGen+Sao+xIT N1: resto	11,291	43,929	1,607
3	N0: Inter N1: Sao+xIT+parser N2: resto	10,363	47,863	1,751
3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	11,350	43,700	1,598

Tabla 21. Mejores resultados obtenidos mediante programa (simulación completa)

ii. Resultados para vídeo *RA_BasketballDrill_832x480_50_qp27.bin*

El vídeo tiene una resolución de 832 x 480 y decodifica 494 imágenes en cada ejecución. El vídeo se muestra en la Figura 49.

Figura 49. Secuencia de vídeo *RA_BasketballDrill_832x480_50_qp27.bin*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	18,804	26,271	1,000
2	N0: Inter+xIT N1: resto	12,383	39,893	1,519
2	N0: Intra+xIT N1: resto	15,757	31,351	1,193
2	N0: Inter+xIT+source+display N1: resto	12,120	40,759	1,551
3	N0: Inter+xIT N1: source+display+merge N2: resto	12,286	40,208	1,531
3	N0: Inter N1: Intra N2: resto	14,211	34,762	1,323

Tabla 22. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	18,804	26,271	1,000
2	N0: Intra+Inter+Sao+SelectCU N1: resto	10,920	45,238	1,722
2	N0: Source+Inter+Intra+Sao+SelectCU+QqGen+Sao N1: resto	10,934	45,180	1,720
2	N0: Inter+Intra+SelectCU+QpGen+Sao N1:resto	10,911	45,275	1,723
3	N0: Inter N1: Sao+xIT+parser N2: resto	10,268	48,111	1,831
3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	10,813	46,922	1,786

Tabla 23. Mejores resultados obtenidos mediante programa (simulación completa)

iii. Resultados para vídeo *LD_RaceHorses_832x480_30_qp27.bin*

El vídeo tiene una resolución de 832 x 480 y decodifica 297 imágenes en cada ejecución. El vídeo se muestra en la Figura 50.

Figura 50. Secuencia de vídeo *LD_RaceHorses_832x480_30_qp27.bin*

Cores	***.xcf***	***time(s)***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	14,910	19,920	1,000
2	N0: Inter+xIT N1: resto	10,390	28,585	1,435
2	N0: Intra+xIT N1: resto	13,692	21,691	1,089
2	N0: Inter+xIT+source+display N1: resto	10,287	28,781	1,445
3	N0: Inter+xIT N1: source+display+merge N2: resto	10,362	28,662	1,439
3	N0: Inter N1: Intra N2: resto	11,779	25,214	1,266

Tabla 24. Mejores resultados obtenidos de forma manual

Cores	***.xcf***	***time(s)***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	14,910	19,920	1,000
2	N0: Parser+Inter+Sao N1: resto	9,490	31,296	1,571
2	N0: Source+Inter+Sao+QqGen+Parser N1: resto	9,480	31,329	1,573
2	N0: Source+Inter+Sao+Parser N1: resto	9,479	31,332	1,573
3	N0: Inter N1: Sao+xIT+parser N2: resto	10,157	29,118	1,462
3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	10,705	27,697	1,390

Tabla 25. Mejores resultados obtenidos mediante programa (simulación completa)

iv. Resultados para vídeo RA_*RaceHorses_832x480_30_qp27.bin*

El vídeo tiene una resolución de 832 x 480 y decodifica 296 imágenes en cada ejecución. El vídeo se muestra en la Figura 51.

Figura 51. Secuencia de vídeo RA_*RaceHorses_832x480_30_qp27.bin*

Cores	***.xcf***	***time(s)***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	13,444	22,017	1,000
2	N0: Inter+xIT N1: resto	9,780	30,266	1,375
2	N0: Intra+xIT N1: resto	12,401	23,869	1,084
2	N0: Inter+xIT+source+display N1: resto	9,610	30,801	1,399
3	N0: Inter+xIT N1: source+display+merge N2: resto	9,725	30,437	1,382
3	N0: Inter N1: Intra N2: resto	10,788	27,438	1,246

Tabla 26. Mejores resultados obtenidos de forma manual

Cores	***.xcf***	***time(s)***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	13,444	22,017	1,000
2	N0: Source+Parser+Inter+Sao+QpGen N1: resto	8,655	34,200	1,553
2	N0: Parser+Intra+QpGen+Sao N1: resto	8,650	34,220	1,554
2	N0: Parser+Inter+QpGen+Sao N1: resto	8,648	34,228	1,555
3	N0: Inter N1: Sao+xIT+parser N2: resto	8,301	35,093	1,594
3	N0: Inter+qpGen+Sao N1: Intra+xIT N2: resto	10,045	29,705	1,349

Tabla 27. Mejores resultados obtenidos mediante programa (simulación completa)

v. Resultados para vídeo *LD_BasketballDrill_832x480_50_qp32.bin*

El vídeo tiene una resolución de 832 x 480 y decodifica 493 imágenes en cada ejecución. El vídeo es el siguiente:

Figura 52. Secuencia de vídeo *LD_BasketballDrill_832x480_50_qp32.bin*

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	18,931	26,042	1,000
2	N0: Inter+xIT N1: resto	13,001	37,920	1,456
2	N0: Intra+xIT N1: resto	15,246	32,336	1,242
2	N0: Inter+xIT+source+display N1: resto	12,637	39,012	1,498
3	N0: Inter+xIT N1: source+display+merge N2: resto	13,695	35,999	1,382
3	N0: Inter N1: Intra N2: resto	14,929	33,023	1,268

Tabla 28. Mejores resultados obtenidos de forma manual

Cores	.xcf	time(s)	FPS	Gain(M)
1	Predeterminado (1 núcleo)	18,931	26,041	1,000
2	N0: Source+Inter+Sao+xIT+generateInfo N1: resto	9,281	53,119	2,040
2	N0: Inter+Sao+xIT+generateInfo N1: resto	9,269	53,188	2,042
2	N0: Source+Inter+Sao+xIT+QpGen N1: resto	9,066	54,379	2,088
3	N0: Inter N1: Sao+xIT+parser N2: resto	10,683	46,148	1,772
3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	9,534	50,024	1,921

Tabla 29. Mejores resultados obtenidos mediante programa (simulación completa)

7.1.7 Conclusiones sobre los resultados del decodificador HEVC

A la vista de los resultados obtenidos, se puede asegurar que para el decodificador HEVC estudiado, los bloques de actores que más peso tienen son las predicciones Inter seguido de los bloques de actores xIT e Intra. Esto se ha podido comprobar con los mejores resultados obtenidos para cada decodificador y con los resultados de las mejores distribuciones posibles.

Para analizar tanto la funcionalidad como la eficacia del programa desarrollado, se va a realizar una comparativa de los mejores resultados de ganancia obtenidos de forma manual frente a los obtenidos mediante el programa, para los diferentes núcleos con la mejor de las distribuciones obtenidas.

Decoder	Secuencia	Core	Gain(M)	Gain(P)	Mejora
Top_mpegh_part2_main_no_md5	ld_BasketballDrill_qp27.bin	2	1,458	1,701	16,67%
Top_mpegh_part2_main_no_md5	ld_BasketballDrill_qp27.bin	3	1,448	1,751	20,93%
Top_mpegh_part2_main_no_md5	ra_BasketballDrill_qp27.bin	2	1,551	1,723	11,09%
Top_mpegh_part2_main_no_md5	ra_BasketballDrill_qp27.bin	3	1,531	1,831	19,60%
Top_mpegh_part2_main_no_md5	ld_BasketballDrill_qp32.bin	2	1,498	2,088	39,39%
Top_mpegh_part2_main_no_md5	ld_BasketballDrill_qp32.bin	3	1,382	2,310	67,15%
Top_mpegh_part2_main_no_md5	ld_RaceHorses_qp27.bin	2	1,445	1,574	8,93%
Top_mpegh_part2_main_no_md5	ld_RaceHorses_qp27.bin	3	1,439	2,069	43,78%
Top_mpegh_part2_main_no_md5	ra_RaceHorses_qp27.bin	2	1,399	1,592	13,80%
Top_mpegh_part2_main_no_md5	ra_RaceHorses_qp27.bin	3	1,382	1,594	15,34%

Tabla 30. Comparación de los resultados de ganancia Decodificador HEVC

Como se puede observar en la tabla anterior, los mejores resultados se obtienen mediantes distribuciones en 3 núcleos frente a 2 núcleos, independientemente del decodificador y vídeo utilizados. Si se hace referencia a todos los resultados obtenidos mediante el decodificador HEVC, se puede afirmar, que el conjunto de actores que tienen más carga, y por lo tanto más relevancia son los que forman la predicción Inter, seguidos por los bloques de actores xIT y la predicción Intra. Realizando una buena distribución de estos actores, se puede conseguir aumentar considerablemente la ganancia obtenida frente a los resultados para 1 núcleo.

También, haciendo uso de los resultados de la Tabla 30, se puede ver la eficacia y la importancia del programa desarrollado. Mediante el programa se consigue en todos los casos una mejoría con respecto a los resultados obtenidos de forma manual por el usuario. Esto quiere decir que, aunque el usuario posea grandes conocimientos sobre el decodificador y realice buenas distribuciones de sus actores sobre los núcleos, el programa prácticamente siempre obtendrá al menos una distribución mejor y más eficiente, sin la necesidad de la intervención del usuario. Este hecho resalta tanto la eficacia como la comodidad de la aplicación desarrollada.

7.2 Resultados mediante simulación agrupada

Este apartado recoge las pruebas y resultados obtenidos mediante la ejecución del programa en el modo “simulación agrupada”, que puede ser seleccionado por el usuario como se ha indicado anteriormente.

Se han realizado pruebas para los decodificadores MPEG 4 parte 2 SP y parte 10 CBP para distribuciones de 2 y 3 núcleos. No se han reflejado los resultados de las simulaciones obtenidos con el decodificador MPEG 4 parte 10 PHP debido a que

prácticamente eran los mismo que para el decodificador anterior CBP. Actualmente se está trabajando para adaptar este método a los nuevos decodificadores como HEVC, en el que es menos eficaz, debido a la gran complejidad del decodificador y de sus actores.

7.2.1 Resultados decodificador MPEG 4

i. Resultados para decodificador parte 2 SP

Se ha utilizado el vídeo *foreman_cif_xvid_384kbps_I_P.m4v* cuyas características ya han sido detalladas en el apartado anterior. Los resultados obtenidos son:

Cores	.xcf	FPS	Gain(M)
1	Predeterminado	104,708	1,000
2	N0: motionY+textureY+motionU+textureU+motionV+textureV N1: resto	197,964	1,891
2	N0: motionY+textureY+motionU+textureU N1: resto	218,593	2,088
2	N0: motionY+textureY+motionV+textureV N1: resto	215,214	2,055
2	N0: motionY+textureY N1: resto	232,995	2,225
3	N0: motionY+textureY N1:motionU,V+textureU,V N2:resto	206,988	1,977
3	N0: motionY+textureY+merger N1:motionU,V+textureU,V N2:resto	191,662	1,830
3	N0: motionY+textureY N1: motionU+textureU N2: resto	198,775	1,898

Tabla 31. Mejores resultados obtenidos mediante la simulación agrupada

ii. Resultados para decodificador parte 10 CBP

Se ha utilizado el vídeo *LS_SVA_D.264* cuyas características ya han sido detalladas en el apartado anterior. Los resultados obtenidos son:

Cores	***.xcf***	***FPS***	***Gain(M)***
1	Predeterminado (1 núcleo)	152,808	1,000
2	N0: Y+U+V N1: resto	359,937	2,355
2	N0: Y+U N1: resto	360,714	2,361
2	N0: Y+V N1: resto	358,911	2,349
2	N0: Y N1: resto	376,866	2,466
2	N0: V+U N1: resto	334,601	2,190
3	N0: merger+Y N1: U N2: resto	269,584	1,764
3	N0: Y+V N1:U+merger N2: resto	287,3906	1,881

Tabla 32. Mejores resultados obtenidos mediante la simulación agrupada

Con este tipo de simulación, se logran disminuir notablemente los tiempos de simulación de cada decodificador, pero se puede llegar a perder eficacia en encontrar las mejores simulaciones. Esto quiere decir que, debido a que se realizan un número menor de simulaciones, se pueden llegar a omitir simulaciones con distribuciones eficaces; no

obstante, de este modo, también se obtienen muy buenos resultados de simulación en mucho menor tiempo.

7.3 Simulación completa vs. Simulación agrupada

En este apartado, se realiza una comparativa entre los dos tipos de simulaciones que puede llevar a cabo el usuario, la simulación completa y la simulación agrupada. Se realizará una comparación de los mejores resultados obtenidos mediante ambas simulaciones y se explicarán las ventajas e inconvenientes de cada una de ellas.

La siguiente Tabla 33, mostrará los resultados obtenidos en ambos casos:

Decoder	Secuencia	Core	G(agrupada)	G(completa)	Mejora
Mpeg part2 SP	foreman_cif_xvid_384kbps_I_P.m4v	2	2,225	2,597	16,72%
Mpeg part2 SP	foreman_cif_xvid_384kbps_I_P.m4v	3	1,977	3,187	61,20%
Mpeg part10 CBP	LS_SVA_D.264	2	2,466	2,756	11,76%
Mpeg part10 CBP	LS_SVA_D.264	3	1,881	3,532	87,77%

Tabla 33. Resultados de ganancia de simulación agrupada vs completa

En primer lugar, resaltar que cualquiera de las dos simulaciones obtiene mejores resultados que las simulaciones realizadas manualmente por el usuario. Una vez dicho esto, se puede observar que la mejora de ganancia obtenida mediante la simulación completa, y por lo tanto de las mejores distribuciones, es elevada frente a la simulación agrupada. Bien es cierto que, la ventaja de la simulación agrupada frente a la simulación completa es la reducción del tiempo de simulación, que puede llegar a reducirse entre un 30% - 50% y aunque los resultados que se obtienen con este tipo de simulación no son tan buenos como en la simulación completa, son bastante razonables y de una calidad aceptable frente a los resultados manuales.

7.4 Tiempos de ejecución

Hay muchos factores a tener en cuenta a la hora de analizar los tiempos de ejecución obtenidos en cada una de las pruebas realizadas. Esto hace que los tiempos de ejecución difieran unos de otros.

- Decodificador: hay que tener en cuenta la complejidad del mismo, el número de actores y la eficiencia a la hora de decodificar el vídeo.
- Vídeo: según el tipo de vídeo que se utilice así como el formato en el que este encapsulado, hay que tener en cuenta los siguientes factores referidos al mismo.
    - Duración, número de imágenes que contiene la secuencia.
    - Cantidad de movimiento: la cantidad de imágenes en movimiento que tenga el vídeo afecta al tiempo de ejecución del mismo. Cuantas más

imágenes en movimiento haya, más complicada será la decodificación y por lo tanto el tiempo de ejecución aumentará.
  - Resolución espacial: El tamaño de reproducción del vídeo afecta al tiempo de ejecución, a medida que aumenta el tamaño del vídeo, aumenta el tiempo de ejecución.
  - Calidad de imagen del vídeo: si se decodifica una imagen con una calidad de imagen baja, el tiempo de ejecución se reduce en comparación con una imagen de calidad aceptable o alta (según la calidad y formato de vídeo aceptado por el decodificador utilizado).
- Número de bloques de actores: A medida que aumenta el número de bloques de actores, el tiempo de simulación aumenta exponencialmente debido al aumento de pruebas a realizar. Esta relación queda definida de la siguiente forma:

$$\text{nº de simulaciones} = 2^{\text{nº bloques de actores}} \text{ para 2 núcleos}$$

$$\text{nº de simulaciones} = 3^{\text{nº bloques de actores}} \text{ para 3 núcleos}$$

$$\text{nº de simulaciones} = nucleos^{\text{nº bloques de actores}}$$

- Distribución de los actores: La distribución de los actores sobre los diferentes núcleos es el principal factor influyente a la hora de analizar los tiempos de ejecución. El programa genera todas las distribuciones posibles para el número de núcleos establecido por el usuario, por lo que se pueden obtener distribuciones muy buenas o en algunos casos muy malas. Si la distribución es buena, se disminuye el tiempo medio de ejecución del vídeo; pero en el caso contrario, si la distribución es mala, el tiempo aumentará de forma considerable.
- Plataforma o sistema operativo: la plataforma o dispositivo sobre el que se ejecute el programa es un factor a tener en cuenta. Hay que tener en cuenta las características tanto hardware como software de la plataforma.

A continuación se analizarán los tiempos medios de ejecución del programa desarrollado para cada uno de los decodificadores estudiados, teniendo en cuenta el número de núcleos sobre los que se ejecuta y la duración media de los vídeos utilizados para cada decodificador. Los tiempos que aparecen a continuación con realizando simulaciones completas.

Cores	Nº de simulaciones	tiempo medio de ejecución
1	1	duración del vídeo
2	128	10 - 12 min
3	16384	1280 - 1536 min // 21,3 - 25,6 h
4	32768	2560 - 3072 min // 42,6 - 51,2 h

Tabla 34. Tiempo medio de ejecución decodificador Mpeg 4 part 2 SP

Cores	Nº de simulaciones	tiempo medio de ejecución
1	1	duración del vídeo
2	256	20 - 25 min
3	65536	5120 - 6400 min // 85,3 - 106,6 h
4	131072	10240 - 12800 min // 170,6 - 213,2 h

Tabla 35. Tiempo medio de ejecución decodificador Mpeg 4 part 10 CBP

Cores	Nº de simulaciones	tiempo medio de ejecución
1	1	duración del vídeo
2	256	20 - 25 min
3	65536	5120 - 6400 min // 85,3 - 106,6 h
4	131072	10240 - 12800 min // 170,6 - 213,2 h

Tabla 36. Tiempo medio de ejecución decodificador Mpeg 4 part 10 PHP

Cores	Nº de simulaciones	tiempo medio de ejecución
1	1	duración del vídeo
2	4096	320 - 384 min // 5,3 - 6,4 h
3	16777216	1310720 - 1572864 min // 21845 - 26214 h
4	33554432	2621440 - 3145728 min // 43690 - 54428 h

Tabla 37. Tiempo medio de ejecución decodificador HEVC

7.4.1 Conclusiones sobre los tiempos de ejecución

A la vista de los resultados obtenidos en el apartado anterior, se puede llegar a la conclusión de que, a medida que aumenta la complejidad del decodificador utilizado (aumento del número de actores), el tiempo que emplea el programa desarrollado para realizar todas las distribuciones posibles aumenta muy considerablemente.

La aplicación desarrollada, ofrece la posibilidad de disminuir estos tiempos de ejecución, eligiendo la opción de simulación agrupada, lo que reduce estos tiempos aproximadamente a la mitad, pero sin realizar todas las comprobaciones de distribuciones posibles, por lo que se pueden obviar algunas de las configuraciones con buenos resultados de simulación.

CAPÍTULO 8. CONCLUSIONES Y FUTUROS TRABAJOS

El presente Proyecto Fin de Carrera, se ha llevado a cabo principalmente para facilitar una de las tareas más ardua y costosa a la hora de estudiar las mejores distribuciones de los actores de cada uno de los decodificadores estudiados y obtener el mejor resultado posible. El programa desarrollado es una aplicación que resulta bastante útil para conocer, analizar y obtener una gran cantidad de resultados y con ellos los diferentes decodificadores.

Este proyecto puede ser de utilidad en futuros proyectos o trabajos que se realicen sobre el campo de RVC. Una de las posibilidades sería integrar y utilizar el programa en diferentes plataformas, como por ejemplo un DSP

Otra posibilidad sería adecuar o mejorar este mismo programa, para hacerlo más eficiente y poder añadirle funciones o características nuevas, como por ejemplo:

- Métodos de análisis y ejecución más rápidos.
- Mejorar la flexibilidad a la hora de realizar las distribuciones.
- Poder analizar varios ficheros de distribución para una misma ejecución
- Almacenar los resultados mediante un fichero Excel.
- Profundizar más en el nivel de análisis y creación de los bloques de actores.
- Realizar un análisis del número de operaciones y realizar una estimación del tiempo que conlleva.

Una mejora bastante relevante que se podría hacer sobre el programa, sería el intentar reducir al máximo los tiempos de ejecución de cada prueba realizada, lo que supondría una gran ventaja al poder obtener todas las distribuciones de los actores en los núcleos de forma más rápida.

Para finalizar, este PFC me ha permitido adentrarme en el mundo de RVC y estudiar sus características y utilidades, lo que me ha resultado muy interesante. Este campo tiene un potencial de desarrollo impresionante, debido a los grandes avances tecnológicos y de mercado, que intentaré seguir de cerca una vez finalizado este PFC.

REFERENCIAS

[1] ISO/IEC 23001-4 o MPEG-B pt 4.

[2] ISO/IEC 23002-4 o MPEG-C pt 4.

[3] Open RVC-CAL Compiler.

Web site: orcc.sourceforge.net

[4] MPEG Reconfigurable Video Coding. Marco Mattaveli, Jörn W. Janneck and Michaël Raulet.
Author manuscript, published in "Handbook of Signal Processing System, Bhattacharyya, Shuura S. and Deprettere, Ed. F. and Leupers, Rainer and Tahala, Jarmo (Ed.) (2010) 43-67".

[5] Cal Actor Language.
Indicaciones de la instalación de herramientas. Metodología de trabajo con el estándar de vídeo reconfigurable RVC. Trabajo fin de Máster ISSSI. Miguel Chavarrías Lapastora.

[6] RVC Standard and MPEG RVC Encoding/Decoding Scenario.

Cal Dataflow Components for an MPEG RVC AVC Baseline encoder. (pdf). Aman-Allah, H. Hanna, E.Maarouf, K. Amer (2009)

Tavars a comprehensive RVC VTL: A CAL Description of an Efficient AVC Baseline Encoder. IEEE International Conference on Image Processing (ICIP 2009).

Jang. E.S, Ohm J Mattavelli. M (January 2008).
Whitepaper on Reconfigurable Video Coding (RVC).
ISO/IEC JTC1/SC29/WG11 document N9586.

[7] Estudio de Vídeo y RVC.

Contribución a las metodologías de optimización del tiempo de ejecución de algoritmos de descodificación de vídeo sobre DSPs.
Tesis Doctoral, Fernando Pescador del Oso.

IEEE transactions on circuits and systems for video technology, vol. 22, no. 12, december 2012. Andrey Norkin, Gisle Bjøntegaard, Arild Fuldseth, Matthias

Narroschke, Masaru Ikeda, Kenneth Andersson, Minhua Zhou, and Geert Van der Auwera.

Sample Adaptive Offset in the HEVC Standard.
Chih-Ming Fu, Elena Alshina, Alexander Alshin, Yu-Wen Huang, Ching-Yeh Chen, and Chia-Yang Tsai, *Member, IEEE,* Chih-Wei Hsu, Shaw-Min Lei, *Fellow, IEEE,* Jeong-Hoon Park, and Woo-Jin Han, *Member, IEEE*

[8] Eclipse.

www.eclipse.org
www.ecured.cu/index.php/Eclipse,_entorno_de_desarrollo_integrado

[9] Graphiti.

www.eclipse.org/graphiti

[10] CMake.

www.cmake.org

[11] GCC.

www.gcc.gnu.org

[12] Ley Amdahl.

Gene Amdahl, "Validity of the single Processor Approach to Achieving Harge-Scale Computing Capabilities", AFIPS Conference Proceedings, (so), pp.483-485, 1967.

[13] Normas ISO/IEC.

www.iso.org

[14] Ventajas de HEVC.

www.configuraequipos.com/actualidad-informatica/5758/hevc-y-h265-formato-y-codec-sucesores-del-h264.

www.panoramaaudiovisual.com/2013/10/07/por-que-hevc-esta-llegando-mas-rapido-de-lo-que-piensa

[15] MPEG Systems technologies Part 4: Codec Configuration Representation 2011.

[16] MPEG Video Technologies Part4: Video Tool Library.

ANEXO 1. Procedimientos de instalación

I. Instalación del entorno de trabajo

Esta sección describirá los pasos a seguir para instalar el entorno de trabajo con todos los programas necesarios, sobre una plataforma PC. El objetivo de este anexo es proporcionar toda la información de las herramientas necesarias para empezar a trabajar con el estándar MPEG RVC-CAL (H.264 y H.265) para el compilador Orcc, como se ha expuesto anteriormente en el desarrollo del trabajo. Debido a esto, es recomendable leer por completo el anexo antes de empezar con la instalación.

a) Es necesario tener instalado un sistema operativo, independientemente de tenerlo instalado directamente en el PC o a través de una maquina virtual. En este caso se ha utilizado Linux Ubuntu versión 10.04.
El procedimiento de instalación de éste es el siguiente:

- Paso 1: Obtener el CD de Ubuntu y colocarlo en la lectora de cd. En la configuración de la BIOS hay que indicar que debe iniciar (bootear) primeramente desde el cd.

- Paso 2: Elegir el idioma de instalación e instalar Ubuntu 10.04.

Figura 53. Paso 2 de instalación de Eclipse

- Paso 3: Elegir la ubicación y la zona horaria:

Figura 54. Paso 3 de instalación de Eclipse

- Paso 4: Seleccionar la distribución del teclado.

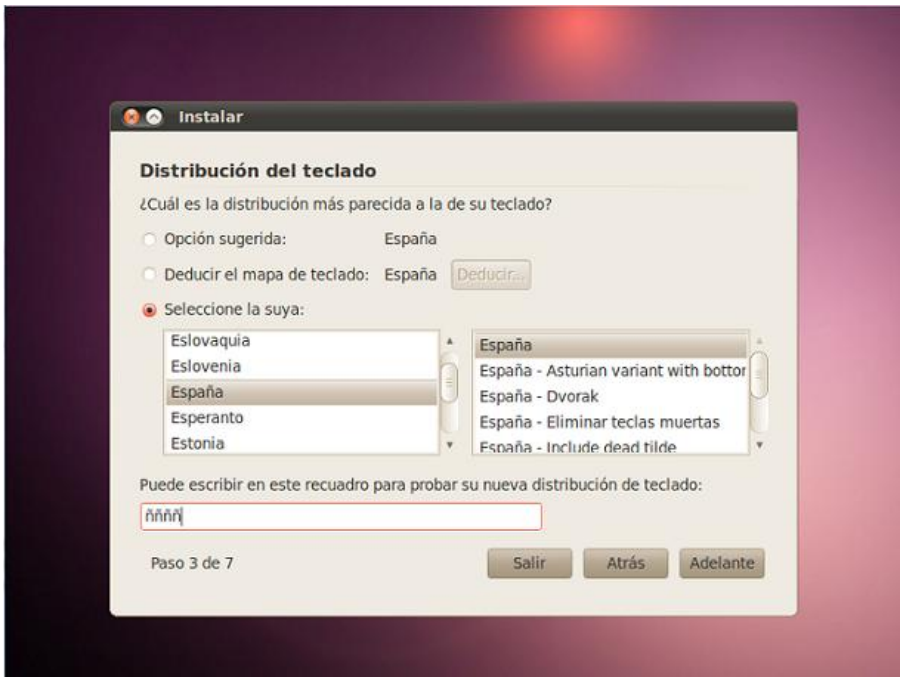


Figura 55. Paso 4 de instalación de Eclipse

- Paso 5: Se elije la forma de particionar el disco duro, bien de forma manual o haciendo que el disco sea una sola partición. En este caso se ha elegido la segunda opción.

Figura 56. Paso 5 de instalación de Eclipse

- Paso 6: Establecer nombre del PC, del usuario así como de la contraseña de inicio de sesión.

- Paso 7: Se muestra una pantalla con un pequeño resumen de las opciones escogidas, y se pulsa instalar.

- Paso 8: Una vez instalado el sistema operativo, habrá que reiniciar el sistema y ya está listo para trabajar sobre él.

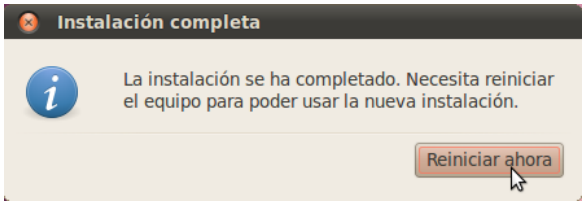


Figura 57. Paso 8 de instalación de Eclipse

b) Instalación de Paquete de Java. Todos los paquetes de instalación están disponibles en la página web de Oracle Referencia con sus respectivos manuales de usuario e instalación dependiendo del sistema operativo utilizado.

- Instalar Java-JDK (Java Development Kit).
- Instalar Java-JRE (Java Runtime Environment).

c) Instalación de Eclipse con las herramientas necesarias.

- Descargar la versión Eclipse Indigo 3.7 Referencia. Hay que tener en cuenta que Eclipse necesita Java tanto para su instalación como su ejecución.

- En el terminal posicionarse en la carpeta donde se ha descargado anteriormente el archivo de eclipse y descomprimirlo mediante el comando *tar –xzvf nombre del archivo*.
- Para comenzar con la instalación del mismo, desde el terminal introducir el comando sudo *apt-get install nombre del archivo.*
- Una vez instalado, se ejecuta el programa y se instalan todos los programas necesarios (Help → Install new software) y se eligen las herramientas a instalar:

    - Eclipse Platform SDK.
    - Eclipse for RCP and RAP Developers.
    - Eclipse SDK.
    - Xtext SDK.

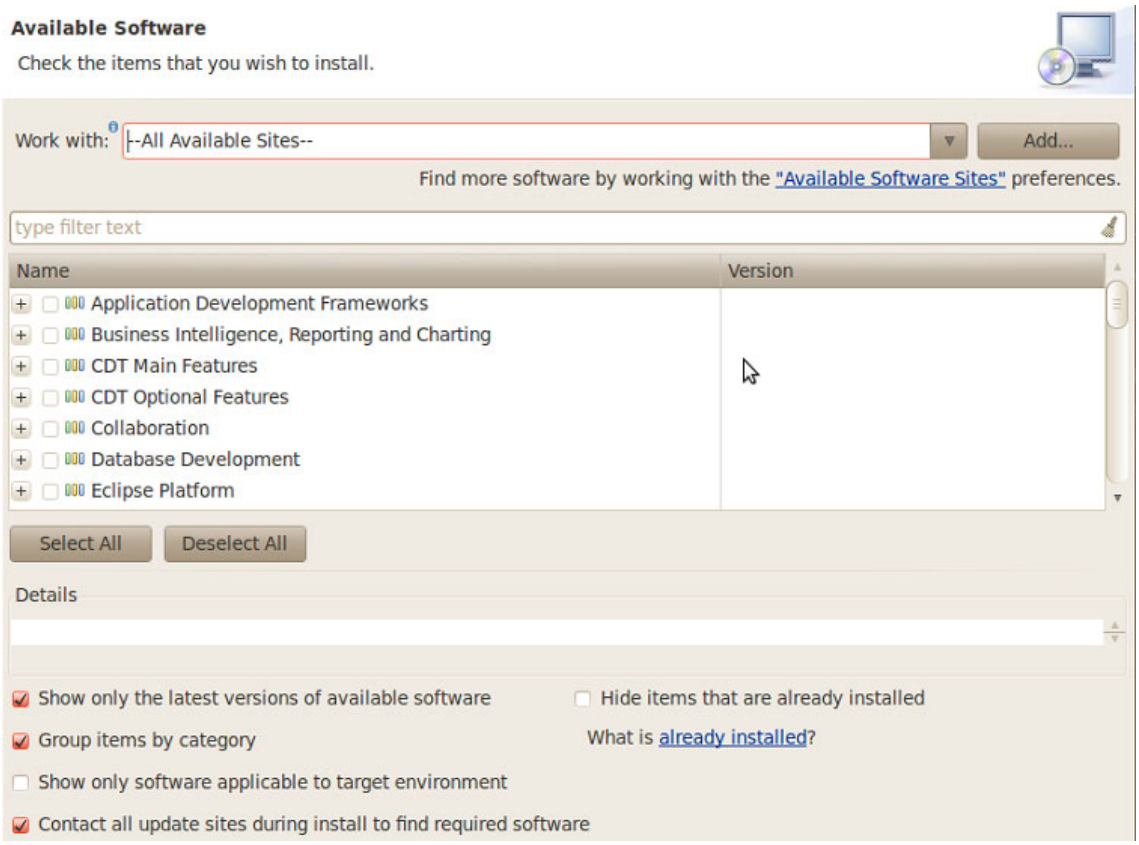


Figura 58. Instalación de las herramientas de Eclipse

Si el desarrollador cree necesario instalar otras herramientas, a demás de éstas, para su programa debería instalarlas en este preciso momento.

- Comprobar si hay nuevas actualizaciones de los programas que tenemos instalados, para ello ir a Help → Check for Updates.

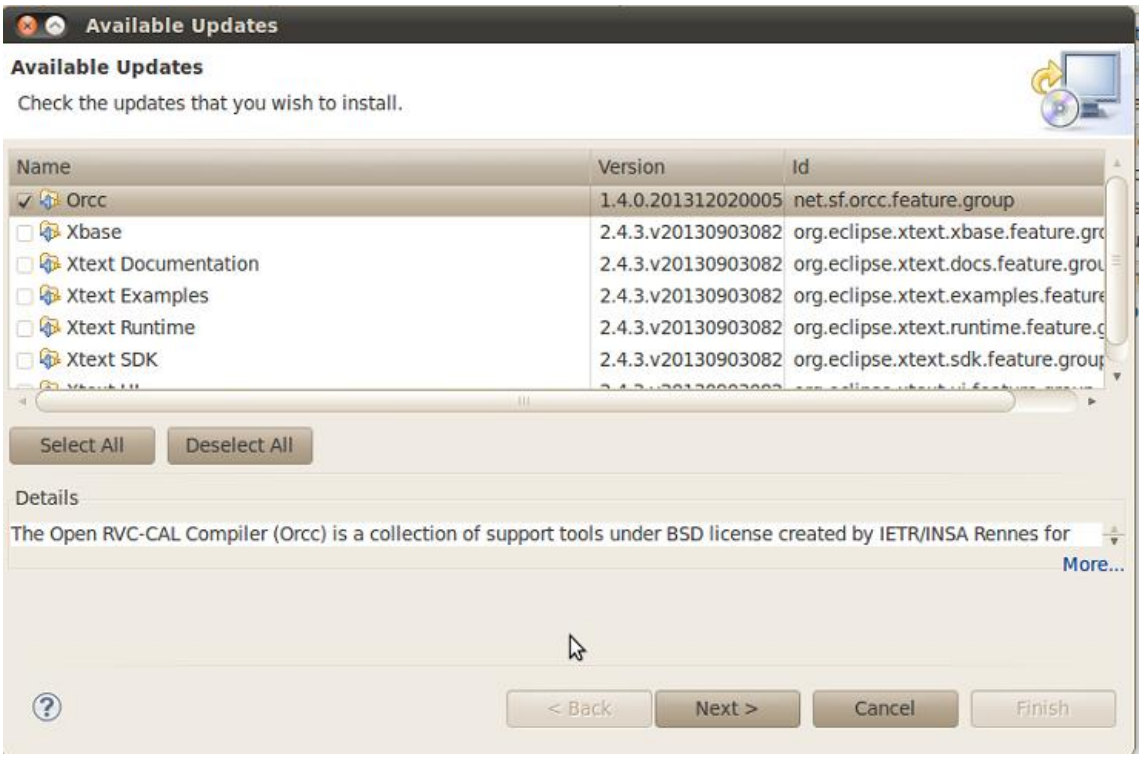


Figura 59. Comprobación de las actualizaciones

- Para comprobar las versiones de los paquetes y herramientas instalados en eclipse basta con ir a Help → About Eclipse → Installation details.

Figura 60. Comprobación de los paquetes y herramientas instalados

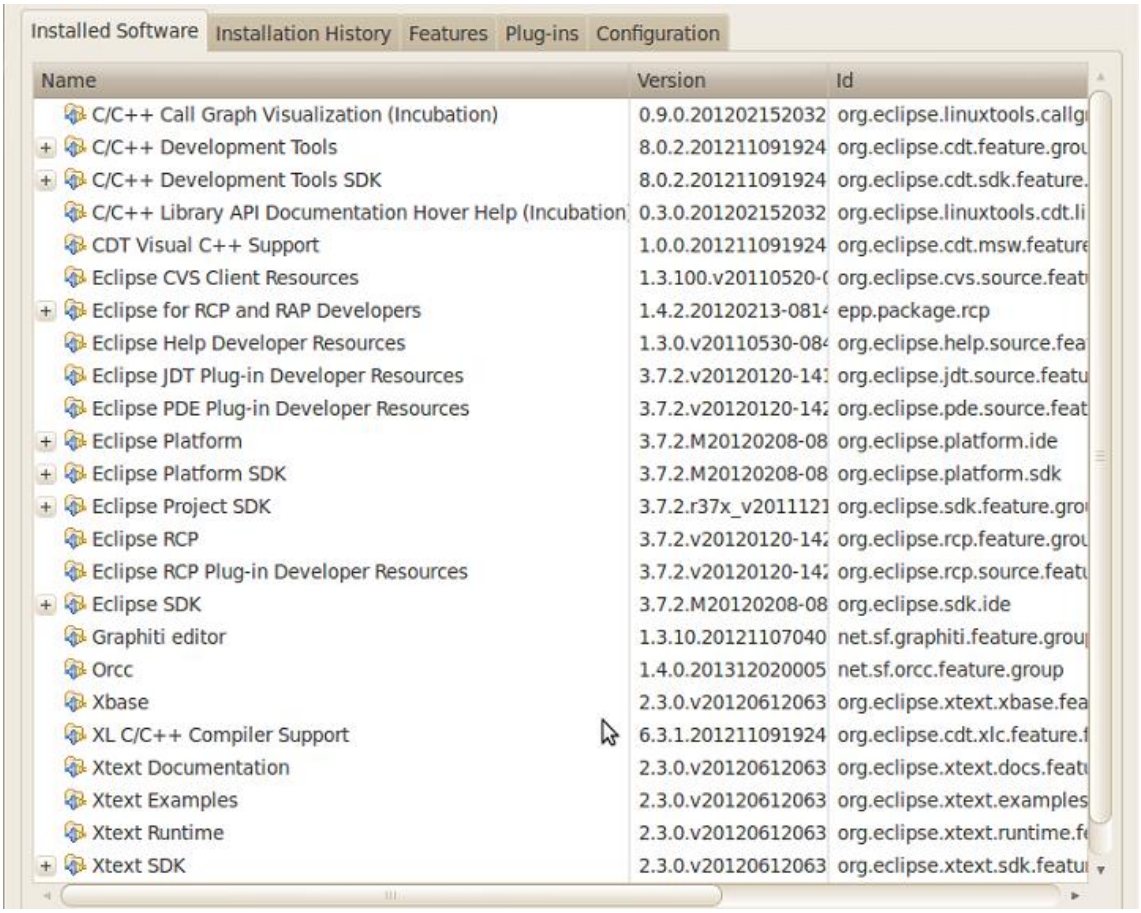


Figura 61. Detalles de instalación

d) Instalación de Orcc y Graphiti. Ambos paquetes pueden ser descargados desde Eclipse como anteriormente se ha especificado. La única diferencia es que hay que añadir el enlace de descarga (http://orcc.sourceforge.net/nightly) para que estos paquetes sean añadidos a la lista de aplicaciones a instalar.

   Consideraciones: durante el proceso de instalación de ambos paquetes, Eclipse mostrará un mensaje indicando que no tienen certificación oficial; esto debe ser ignorado y aceptar continuar con la instalación.

e) Instalar SVN-Workbench a través de la aplicación de gestión de paquetes de Ubuntu denominada Synaptics.

f) Instalar CMake desde Synaptics o desde la página web oficial Referencia.

II. Obtención del decodificador

En este apartado se va a explicar los pasos a seguir para obtener cualquiera de los decodificadores utilizados en el desarrollo del programa.

a) Obtención de los archivos del decodificador.

- Se descargan todos los archivos de los proyectos que se vayan a utilizar tanto para el estudio como para la obtención de los archivos del decodificador a través del enlace http://sourceforge.net/projects/orcc/files o del enlace http://github.com/orcc/orc-apps. A través de estos enlaces se podrá descargar el archivo comprimido denominado *orc-apps.master.zip*. Una vez descargado, se descomprimirá en una carpeta para poder realizar el siguiente paso.

- Se ejecuta Eclipse y se selecciona Eclipse → Import → General → Existing Projects into Workspace (se elige la carpeta donde se ha descargado previamente los archivos).

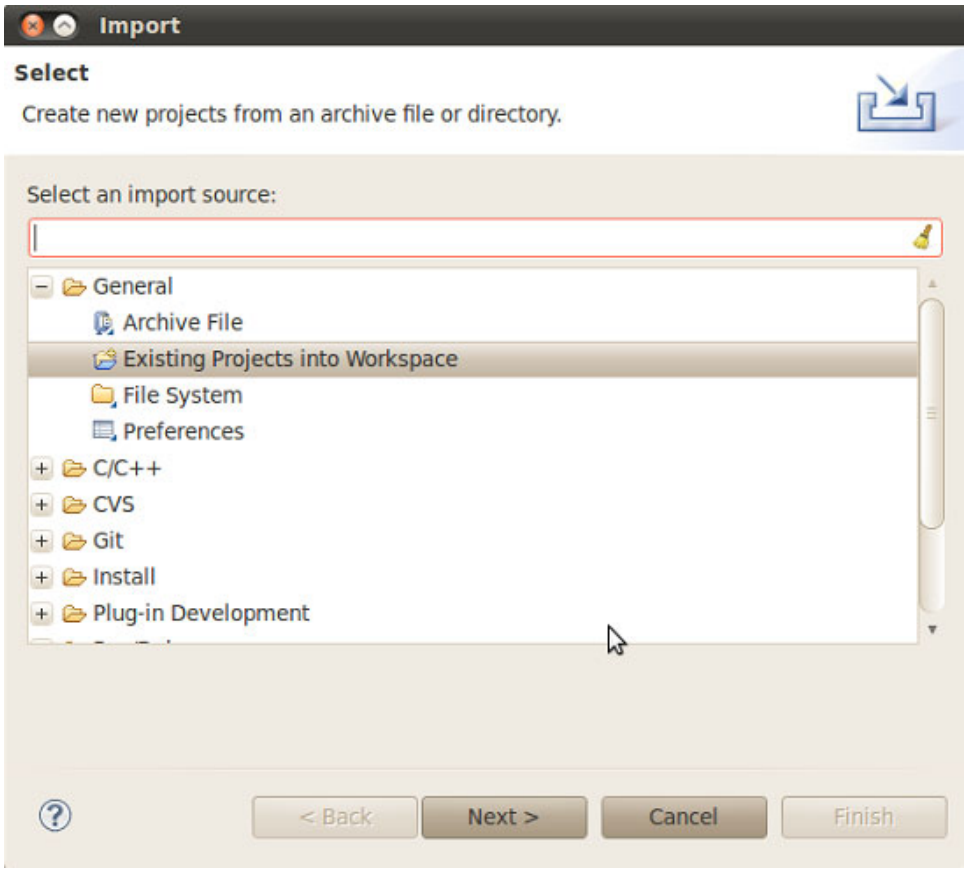


Figura 62. Importar proyectos a entorno de trabajo de Eclipse

- Una vez generado el Workspace, se selecciona el fichero con la extensión .top que se quiere compilar, y se pulsa botón derecho del ratón → Run as → Orcc Compilation o Run Configurations (configuración manual).

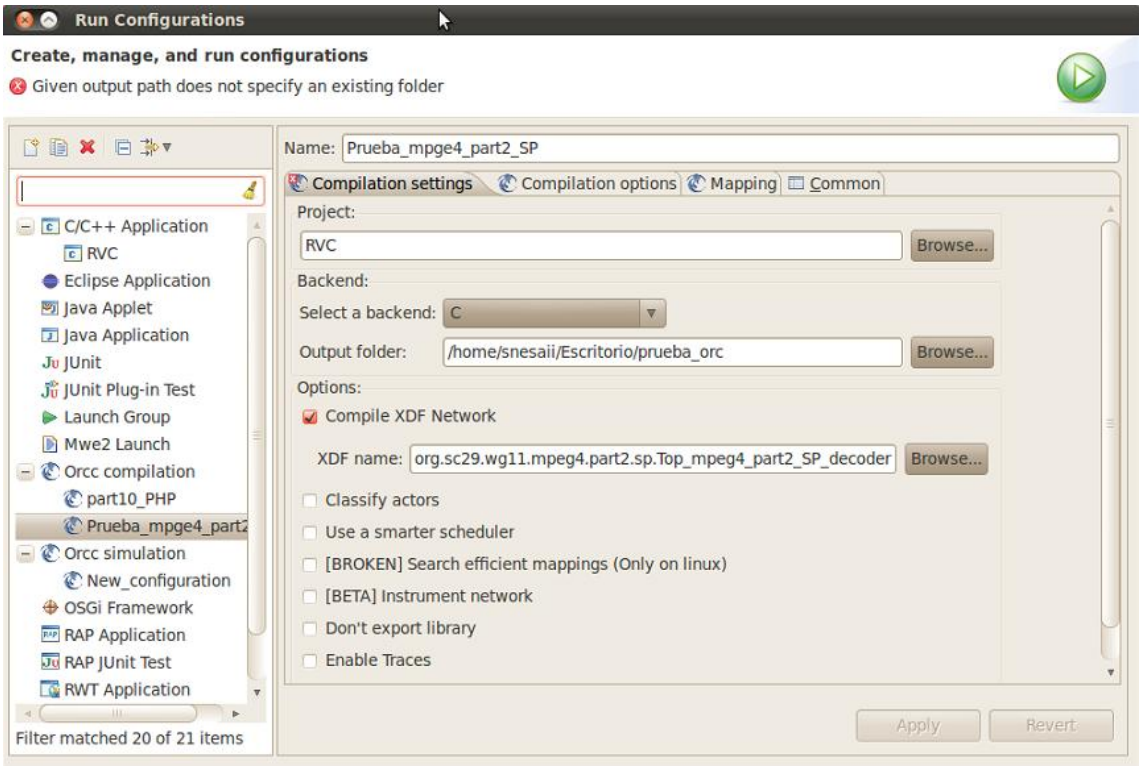


Figura 63. Compilacion del proyecto

Una vez hecho esto, ya han sido generados los archivos que especifican el decodificador.

b) Antes de realizar los pasos para la obtención del decodificador, hay que asegurarse de tener instalados los paquetes de librerías y compilación de aplicación. Para ello hay que seguir estos pasos para su descarga desde el terminal de linux:

- Paquete *Build-essential*: ejecutar el comando *sudo apt-get install build-essential*.
- Librerías SDL: ejecutar el commando *sudo apt-get install libsdl1.2-dev*.
- Imágenes SDL: ejecutar el comando *sudo apt-get install libsdl-image1.2 libsdl-image1.2-dev.*

c) Obtención del ejecutable del decodificador.

- Se ejecuta la aplicación CMake y se compilan los archivos generados anteriormente mediante el fichero *.bat generado.
- Para ello se selecciona para los archivos fuente (*where is the source code*) la carpeta …/srv y para el directorio de archivos (*where to build the binaries*) la carpeta …/build.
- Se pulsa sobre Configure, para obtener todos los parámetros y librerías necesarios.
- Si apareciese algún problema, seleccionar File → Dele Caché, para borrar cualquier tipo de configuración anterior y volver a pulsar nuevamente Configure.
- A continuación se añade una línea con el nombre CMAKE_BUILD_TYPE (si no aparece al realizar Configure) y se establece el valor RELEASE.
- Se pulsa nuevamente sobre Configure y posteriormente sobre Generate para obtener el decodificador elegido.

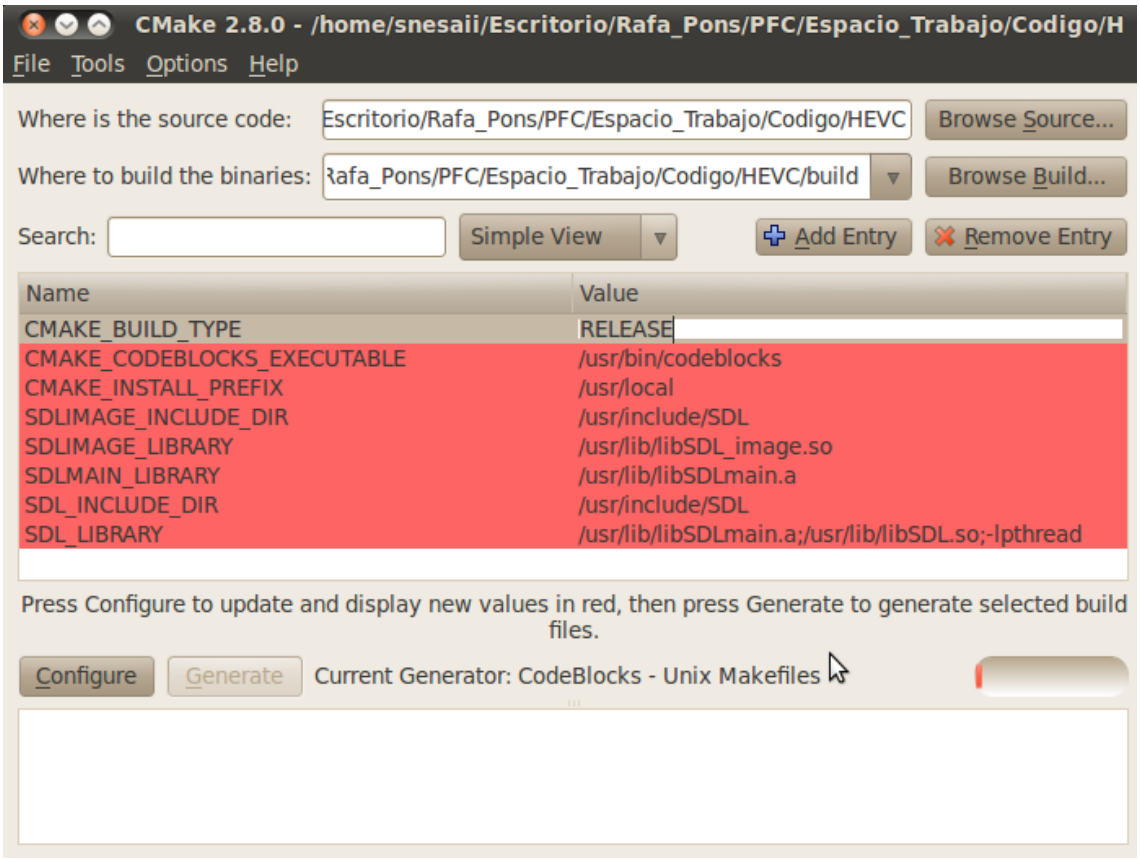


Figura 64. Generación de archivos mediante CMake

*Consideración: Es necesario copiar en la misma carpeta donde se encuentra el ejecutable del decodificador *.exe el archivo de librería SDL.dll para su correcto funcionamiento.*

ANEXO 2. Distribución de los actores

En este anexo, se mostrará de forma detallada la composición de un fichero *.xcf. Para ello se utilizará el archivo de distribución del decodificador MPEG parte 2 SP, debido a que es el decodificador más sencillo de los que se han utilizado. No obstante, la única diferencia que existe entre los archivos de distribución de los diferentes decodificadores, es que los actores de cada decodificador varían. Cuanto más complejo es el decodificador, más complejos son sus actores, y más difícil es obtener distribuciones óptimas.

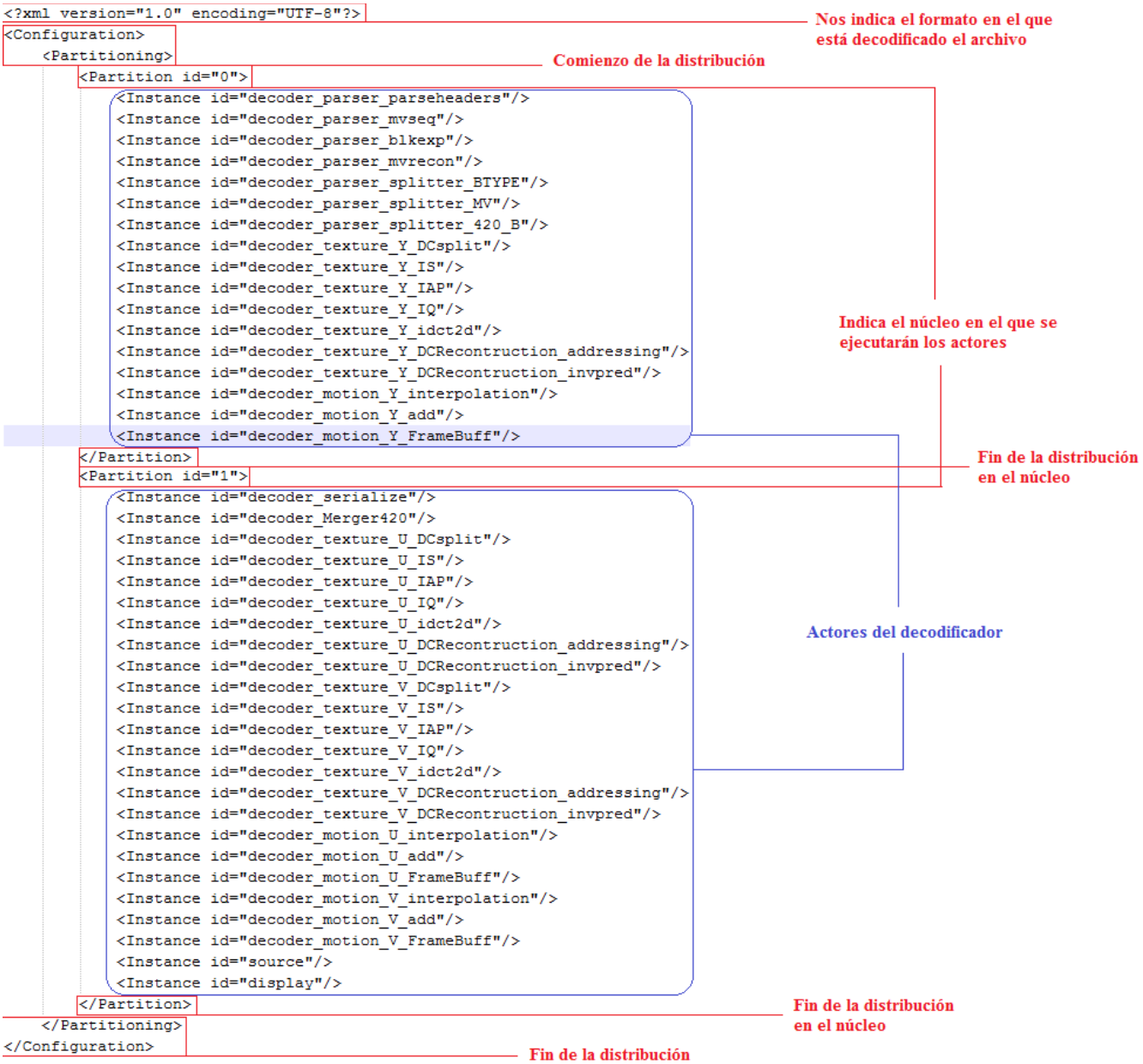


Figura 65. Archivo de distribución *.xcf

Para la realización de la distribución de los actores, se han dividido los actores en bloques de la misma familia. Para entender correctamente esta agrupación se van a analizar los actores correspondientes al decodificador Mpeg 4 part 10 CBP. El número

de actores de este decodificador es de 94, y la agrupación de los mismos, se distribuyen en 9 bloques.

```
<?xml version="1.0" encoding="UTF-8"?>
<Configuration>
    <Partitioning>
        <Partition id="0">
            <Instance id="AVCDecoder_Syn_Parser_Algo_Synp"/>
            <Instance id="AVCDecoder_Syn_Parser_Algo_Parser_IPCM"/>                    Bloque Parser
            <Instance id="AVCDecoder_Syn_Parser_IntraPredSplit"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MMCO"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_RefList0"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_Mgnt_InterPred"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MgntDeblockingFilter"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvL0_Reconstr"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_RefIdxL0ToFrameNum"/>
            <Instance id="AVCDecoder_Syn_Parser_Generate_Inter_Info_MVReconstruct_MvComponentReord"/>
            <Instance id="AVCDecoder_Decoding_Y_DecodedPictureBuffer"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Select"/>                  Bloque Luminancia Y
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Deblocking_Filter"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_DemuxParserInfos"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Mgnt_Intra_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Algo_IntraPred_LUMA_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Add_Clip"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_16x16_Buffer_Neighbour_FullMb"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Mgnt_Intra4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_IntraPred_LUMA_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Add_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Merge_4x4_to_16x16_norasterscan"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Algo_Split_16x16_to_4x4_norasterscan"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Intra_4x4_Buffer_Neighbour_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Algo_Interp_SeparableSixTapQuarterPelAVC"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Add_Clip"/>
            <Instance id="AVCDecoder_Decoding_Y_PredictionY_Inter_Algo_Interp_Reord"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_Algo_Merge_4x4_to_16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_IS_Zigzag_4x4_DC"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_IHT1d_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DC_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_IHT1d_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_Scaling"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DC_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_dcr_Algo_DCR_Hadamard_LUMA_Reordering"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Algo_IS_Zigzag_4x4_AC"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Mgnt_IQ_INTRA16x16"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_Algo_IQ_QSAndSLAndIDCTScaler_4x4"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_1d_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_1d_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decoding_Y_ResidualY_FUN_IS_IQ_IT_L_IT4x4_Algo_IT4x4_Addshift"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Select_chroma"/>             Bloque Crominancia U
            <Instance id="AVCDecoder_Decode_U_PredictionC_Deblocking_Filter"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_DemuxParserInfos"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Algo_IntraPred_CHROMA"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Algo_Add"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Buffer_Neighbour_FullMb"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Intra8x8_Mgnt_Intra_8x8"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Interp_EighthPelBilinear"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Add"/>
            <Instance id="AVCDecoder_Decode_U_PredictionC_Inter_Algo_Interp_Reord"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_Algo_DCR_Hadamard_CHROMA"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_Algo_Merge_4x4_to_8x8"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Algo_IS_Zigzag_4x4"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Mgnt_IQ_Chroma"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_Algo_IQ_QSAndSLAndIDCTScaler_4x4"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_0"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_0"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_1d_1"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_Transpose4x4_1"/>
            <Instance id="AVCDecoder_Decode_U_ResidualC_IS_IQ_IT_C_IT4x4_Algo_IT4x4_Addshift"/>
```

Figura 66. Bloques de actores del decodificador Mpeg 4 parte 10 PHP

ANEXO 3. Resultados obtenidos manualmente

Este anexo recoge todos los resultados obtenidos durante el estudio de los diferentes decodificadores con los que se ha trabajado en este proyecto. Estos resultados son una pequeña muestra de los resultados obtenidos durante todo el desarrollo del proyecto, debido a que se han obtenido una cantidad muy elevado de resultados distintos. En las siguientes tablas, aparecen una batería de resultados que han sido obtenidos mediante la ejecución del programa desarrollado estableciendo como tipo de simulación la completa.

Los resultados se organizarán teniendo en cuenta el decodificador, el vídeo y la distribución utilizada.

I. Decodificador Mpeg4_part2_SP

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
foreman_cif_xvid_384kbps_I_P.m4v	1	Predeterminado	1525	17,728	86,022	1,000
foreman_cif_xvid_384kbps_I_P.m4v	2	Luminancia Y	2170	16,414	132,204	1,537
foreman_cif_xvid_384kbps_I_P.m4v	2	Merger+Y	2164	15,796	136,997	1,593
foreman_cif_xvid_384kbps_I_P.m4v	2	Parser+Y	2166	15,723	137,760	1,601
foreman_cif_xvid_384kbps_I_P.m4v	2	Y,U	2743	17,320	158,372	1,841
foreman_cif_xvid_384kbps_I_P.m4v	2	Y,V	2725	17,472	155,964	1,813
foreman_cif_xvid_384kbps_I_P.m4v	2	Parser+U+V	2431	17,271	140,756	1,636
foreman_cif_xvid_384kbps_I_P.m4v	2	Y+U+Merger	2438	18,193	134,008	1,558
foreman_cif_xvid_384kbps_I_P.m4v	3	Parser+Y+resto	3026	17,565	172,864	2,010
foreman_cif_xvid_384kbps_I_P.m4v	3	Parser,Merger+Y+resto	2762	17,049	162,810	1,893
foreman_cif_xvid_384kbps_I_P.m4v	3	Parser,Y+Merger+resto	1802	16,846	108,165	1,257
foreman_cif_xvid_384kbps_I_P.m4v	3	Y,U+Parser,V+resto	2145	16,522	129,827	1,509
foreman_cif_xvid_384kbps_I_P.m4v	3	Parser,Merger,U+Y+resto	2442	16,512	147,892	1,719
foreman_cif_xvid_384kbps_I_P.m4v	3	Parser+Y,Merger,U+resto	2117	18,179	117,278	1,363

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
chapeau_melon_PAL_IP_track1.avi	1	Predeterminado	513	23,170	22,141	1,000
chapeau_melon_PAL_IP_track1.avi	2	Luminancia Y	512	15,625	32,768	1,480
chapeau_melon_PAL_IP_track1.avi	2	Merger+Y	517	15,605	33,130	1,496
chapeau_melon_PAL_IP_track1.avi	2	Parser+Y	545	15,715	34,680	1,566
chapeau_melon_PAL_IP_track1.avi	2	Y,U	773	19,529	39,582	1,788
chapeau_melon_PAL_IP_track1.avi	2	Y,V	763	19,534	39,060	1,764
chapeau_melon_PAL_IP_track1.avi	2	Parser+U+V	761	20,608	36,927	1,668
chapeau_melon_PAL_IP_track1.avi	2	Y+U+Merger	762	22,381	34,047	1,538
chapeau_melon_PAL_IP_track1.avi	3	Parser+Y+resto	776	20,380	54,161	2,446
chapeau_melon_PAL_IP_track1.avi	3	Parser,Merger+Y+resto	769	20,437	44,627	2,016
chapeau_melon_PAL_IP_track1.avi	3	Parser,Y+Merger+resto	517	21,754	28,122	1,270
chapeau_melon_PAL_IP_track1.avi	3	Y,U+Parser,V+resto	549	16,275	33,733	1,524
chapeau_melon_PAL_IP_track1.avi	3	Parser,Merger,U+Y+resto	829	20,397	40,643	1,836
chapeau_melon_PAL_IP_track1.avi	3	Parser+Y,Merger,U+resto	536	20,015	26,780	1,210

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
5_element_xvid_intra.m4v	1	Predeterminado	11262	20,341	553,660	1,000
5_element_xvid_intra.m4v	2	Luminancia Y	11876	20,399	582,185	1,052
5_element_xvid_intra.m4v	2	Merger+Y	10193	20,341	501,106	0,905
5_element_xvid_intra.m4v	2	Parser+Y	13616	20,307	670,508	1,211
5_element_xvid_intra.m4v	2	Y,U	13412	20,640	649,806	1,174
5_element_xvid_intra.m4v	2	Y,V	12843	20,307	632,442	1,142
5_element_xvid_intra.m4v	2	Parser+U+V	12965	20,478	633,118	1,144
5_element_xvid_intra.m4v	2	Y+U+Merger	12656	20,352	621,855	1,123
5_element_xvid_intra.m4v	3	Parser+Y+resto	17730	21,937	808,224	1,460
5_element_xvid_intra.m4v	3	Parser,Merger+Y+resto	16676	20,436	816,011	1,474
5_element_xvid_intra.m4v	3	Parser,Y+Merger+resto	7057	20,311	347,447	0,628
5_element_xvid_intra.m4v	3	Y,U+Parser,V+resto	11766	20,458	575,130	1,039
5_element_xvid_intra.m4v	3	Parser,Merger,U+Y+resto	13853	20,341	681,038	1,230
5_element_xvid_intra.m4v	3	Parser+Y,Merger,U+resto	14149	20,337	695,727	1,257

II. Decodificador Mpeg4_part10_CBP

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
LS_SVA_D.264	1	Predeterminado	3443	26,057	132,133	1,000
LS_SVA_D.264	2	Luminancia Y	3467	16,860	205,635	1,556
LS_SVA_D.264	2	Merger+Y	3462	17,100	202,456	1,532
LS_SVA_D.264	2	Parser+Y	3503	14,653	2390,637	18,093
LS_SVA_D.264	2	Y,U	3590	15,375	233,496	1,767
LS_SVA_D.264	2	Y,V	3587	15,401	232,907	1,763
LS_SVA_D.264	2	Parser+U+V	3381	15,335	220,476	1,669
LS_SVA_D.264	2	Y+U+Merger	3485	16,953	205,568	1,556
LS_SVA_D.264	3	Parser+Y+resto	3531	15,345	230,108	1,741
LS_SVA_D.264	3	Parser,Merger+Y+resto	3657	15,291	239,160	1,810
LS_SVA_D.264	3	Parser,Y+Merger+resto	3520	16,090	218,769	1,656
LS_SVA_D.264	3	Y,U+Parser,V+resto	3978	15,718	253,086	1,915
LS_SVA_D.264	3	Parser,Merger,U+Y+resto	4958	15,572	318,392	2,410
LS_SVA_D.264	3	Parser+Y,Merger,U+resto	3449	18,952	181,986	1,377

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
HCBP1_HHI_A.264	1	Predeterminado	507	20,904	242,537	1,000
HCBP1_HHI_A.264	2	Luminancia Y	761	17,577	432,952	1,785
HCBP1_HHI_A.264	2	Merger+Y	762	17,922	425,176	1,753
HCBP1_HHI_A.264	2	Parser+Y	787	19,707	399,350	1,647
HCBP1_HHI_A.264	2	Y,U	761	16,959	448,729	1,850
HCBP1_HHI_A.264	2	Y,V	758	19,144	395,947	1,633
HCBP1_HHI_A.264	2	Parser+U+V	1013	21,983	460,811	1,900
HCBP1_HHI_A.264	2	Y+U+Merger	782	18,253	428,423	1,766
HCBP1_HHI_A.264	3	Parser+Y+resto	1027	18,404	586,671	2,419
HCBP1_HHI_A.264	3	Parser,Merger+Y+resto	1015	18,357	579,780	2,390
HCBP1_HHI_A.264	3	Parser,Y+Merger+resto	759	18,777	404,218	1,667
HCBP1_HHI_A.264	3	Y,U+Parser,V+resto	763	18,591	410,414	1,692
HCBP1_HHI_A.264	3	Parser,Merger,U+Y+resto	1306	20,539	635,863	2,622
HCBP1_HHI_A.264	3	Parser+Y,Merger,U+resto	578	16,780	363,253	1,498

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
HCBP2_HHI_A.264	1	si	760	22,909	331,747	1,000
HCBP2_HHI_A.264	2	Luminancia Y	863	15,728	548,703	1,654
HCBP2_HHI_A.264	2	Merger+Y	833	15,701	530,539	1,599
HCBP2_HHI_A.264	2	Parser+Y	924	16,010	577,139	1,740
HCBP2_HHI_A.264	2	Y,U	863	15,618	552,568	1,666
HCBP2_HHI_A.264	2	Y,V	848	15,680	540,816	1,630
HCBP2_HHI_A.264	2	Parser+U+V	857	15,609	549,042	1,655
HCBP2_HHI_A.264	2	Y+U+Merger	815	16,010	509,057	1,534
HCBP2_HHI_A.264	3	Parser+Y+resto	1002	16,533	606,061	1,827
HCBP2_HHI_A.264	3	Parser,Merger+Y+resto	1389	20,731	670,011	2,020
HCBP2_HHI_A.264	3	Parser,Y+Merger+resto	1163	20,454	568,593	1,714
HCBP2_HHI_A.264	3	Y,U+Parser,V+resto	846	16,502	512,665	1,545
HCBP2_HHI_A.264	3	Parser,Merger,U+Y+resto	1609	20,831	779,330	2,349
HCBP2_HHI_A.264	3	Parser+Y,Merger,U+resto	768	16,848	455,840	1,374

III. Decodificador Mpeg4_Part10_PHP

LS_SVA_D.264	2	Merger	3322	32,950	100,819	0,955
LS_SVA_D.264	2	Luminancia Y	3471	20,377	170,339	1,613
LS_SVA_D.264	2	Crominancia blue U	3440	32,053	107,322	1,016
LS_SVA_D.264	2	Crominacia red V	3425	32,261	106,165	1,005
LS_SVA_D.264	2	Merger+Parser	3440	33,596	102,393	0,970
LS_SVA_D.264	2	Merger+Y	3451	20,774	166,121	1,573
LS_SVA_D.264	2	Parser+Y	3474	20,047	173,293	1,641
LS_SVA_D.264	2	U,V	3454	26,007	132,810	1,258
LS_SVA_D.264	2	Y,U	3470	18,884	183,753	1,740
LS_SVA_D.264	2	Y,V	3428	18,786	182,476	1,728
LS_SVA_D.264	2	Parser+U+Merger	3448	26,345	130,879	1,239
LS_SVA_D.264	2	Parser+V+Merger	3428	26,385	129,922	1,230
LS_SVA_D.264	2	Parser+U+V	3484	19,609	177,674	1,682
LS_SVA_D.264	2	Y+U+Merger	3470	20,179	171,961	1,628
LS_SVA_D.264	2	Y+U+V	3457	23,092	149,706	1,418
LS_SVA_D.264	2	Parser+Merger+Source+Display	3456	24,775	139,495	1,321
LS_SVA_D.264	3	Parser+Merger+resto	3440	35,723	162,965	1,543
LS_SVA_D.264	3	Parser+Y+resto	3490	16,669	209,371	1,983
LS_SVA_D.264	3	Merger+Y+resto	3467	23,990	144,519	1,369
LS_SVA_D.264	3	U+V+resto	3451	29,883	115,484	1,094
LS_SVA_D.264	3	Parser,Merger+Y+resto	3511	17,024	206,238	1,953
LS_SVA_D.264	3	Parser,Y+Merger+resto	3896	22,468	173,402	1,642
LS_SVA_D.264	3	Y,U+Parser,V+resto	3564	19,722	180,712	1,711
LS_SVA_D.264	3	Parser,Merger,U+Y+resto	5095	21,251	239,753	2,270
LS_SVA_D.264	3	Parser+Y,Merger,U+resto	3464	22,534	153,723	1,456

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
HCMP1_HHI_A.264	1	Predeterminado	507	32,988	15,369	1,000
HCMP1_HHI_A.264	2	Luminancia Y	509	19,089	26,665	1,735
HCMP1_HHI_A.264	2	Merger+Y	509	19,029	26,749	1,740
HCMP1_HHI_A.264	2	Parser+Y	508	23,093	21,998	1,431
HCMP1_HHI_A.264	2	Y,U	513	20,322	25,244	1,642
HCMP1_HHI_A.264	2	Y,V	508	20,644	24,608	1,601
HCMP1_HHI_A.264	2	Parser+U+V	512	17,281	29,628	1,928
HCMP1_HHI_A.264	2	Y+U+Merger	509	20,753	24,527	1,596
HCMP1_HHI_A.264	3	Parser+Y+resto	763	20,881	36,493	2,374
HCMP1_HHI_A.264	3	Parser,Merger+Y+resto	761	19,129	39,783	2,588
HCMP1_HHI_A.264	3	Parser,Y+Merger+resto	502	23,497	21,364	1,390
HCMP1_HHI_A.264	3	Y,U+Parser,V+resto	509	20,799	24,472	1,592
HCMP1_HHI_A.264	3	Parser,Merger,U+Y+resto	832	23,648	35,183	2,289
HCMP1_HHI_A.264	3	Parser+Y,Merger,U+resto	507	22,893	22,543	1,467

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
HCBP2_HHI_A.264	1	Predeterminado	758	27,307	277,585	1,000
HCBP2_HHI_A.264	2	Luminancia Y	764	16,842	453,628	1,634
HCBP2_HHI_A.264	2	Merger+Y	762	16,891	451,128	1,625
HCBP2_HHI_A.264	2	Parser+Y	764	16,916	451,643	1,627
HCBP2_HHI_A.264	2	Y,U	763	18,376	415,216	1,496
HCBP2_HHI_A.264	2	Y,V	763	15,944	478,550	1,724
HCBP2_HHI_A.264	2	Parser+U+V	772	16,283	474,114	1,708
HCBP2_HHI_A.264	2	Y+U+Merger	762	17,271	441,202	1,589
HCBP2_HHI_A.264	3	Parser+Y+resto	1018	16,804	605,808	2,182
HCBP2_HHI_A.264	3	Parser,Merger+Y+resto	1019	16,725	609,268	2,195
HCBP2_HHI_A.264	3	Parser,Y+Merger+resto	765	18,525	412,955	1,488
HCBP2_HHI_A.264	3	Y,U+Parser,V+resto	778	15,674	496,363	1,788
HCBP2_HHI_A.264	3	Parser,Merger,U+Y+resto	1151	17,228	668,098	2,407
HCBP2_HHI_A.264	3	Parser+Y,Merger,U+resto	763	19,310	395,132	1,423

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
HCBP1_HHI_A.264	1	si	505	25,080	20,136	1,000
HCBP1_HHI_A.264	2	Luminancia Y	510	13,676	37,292	1,852
HCBP1_HHI_A.264	2	Merger+Y	510	14,082	36,216	1,799
HCBP1_HHI_A.264	2	Parser+Y	507	16,710	30,341	1,507
HCBP1_HHI_A.264	2	Y,U	528	16,064	32,869	1,632
HCBP1_HHI_A.264	2	Y,V	578	15,399	37,535	1,864
HCBP1_HHI_A.264	2	Parser+U+V	596	15,566	38,289	1,902
HCBP1_HHI_A.264	2	Y+U+Merger	565	15,450	36,570	1,816
HCBP1_HHI_A.264	3	Parser+Y+resto	781	15,649	49,907	2,479
HCBP1_HHI_A.264	3	Parser,Merger+Y+resto	781	15,561	50,190	2,493
HCBP1_HHI_A.264	3	Parser,Y+Merger+resto	509	17,909	28,421	1,412
HCBP1_HHI_A.264	3	Y,U+Parser,V+resto	601	15,529	38,702	1,922
HCBP1_HHI_A.264	3	Parser,Merger,U+Y+resto	834	15,423	54,075	2,686
HCBP1_HHI_A.264	3	Parser+Y,Merger,U+resto	507	17,498	28,975	1,439

IV. Decodificador HEVC

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
ld_BasketballDrill_qp27	1	Predeterminado	496	21,305	23,281	1,000
ld_BasketballDrill_qp27	2	N0: Inter+Sao+xIT N1: resto	496	11,329	43,781	1,881
ld_BasketballDrill_qp27	2	N0:Inter+QpGen+Sao+xIT N1:resto	496	11,31	43,859	1,884
ld_BasketballDrill_qp27	2	N0:Source+Inter+Sao+xIT N1:resto	496	11,325	43,797	1,881
ld_BasketballDrill_qp27	2	N0:Parser+Inter+Sao+G_Info N1:resto	496	11,353	43,689	1,877
ld_BasketballDrill_qp27	2	N0:Inter+Intra N1:resto	496	17,568	28,233	1,213
ld_BasketballDrill_qp27	2	N0:Inter N1:resto	496	18,198	27,256	1,171
ld_BasketballDrill_qp27	2	N0:Intra N1:resto	496	22,148	22,395	0,962
ld_BasketballDrill_qp27	3	N0: Inter N1: Sao+xIT+parser N2: resto	496	13,442	36,899	1,585
ld_BasketballDrill_qp27	3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	496	13,599	36,473	1,567
ld_BasketballDrill_qp27	3	N0:Inter+Sao+qpGen N1:Intra+xIT N2:resto	496	16,981	29,209	0,667
ld_BasketballDrill_qp27	3	N0:Inter+xIT N1:source+display+merger N2:resto	496	20,751	23,902	1,027
ld_BasketballDrill_qp27	3	N0:Inter+Intra N1: qpGen+Sao N2:resto	496	16,773	29,571	1,270
ra_BasketballDrill_qp27	1	Predeterminado	494	21,09	23,418	1
ra_BasketballDrill_qp27	2	N0:Inter+Intra+Sao+Select N1:resto	494	11,05	44,722	1,9097
ra_BasketballDrill_qp27	2	N0:Inter+Intra+Sao+Source+display N1:resto	494	11,16	44,249	1,8896
ra_BasketballDrill_qp27	2	N0:Inter+Intra+QpGen+Sao N1:resto	494	10,91	45,275	1,9334
ra_BasketballDrill_qp27	2	N0:Inter+QpGen+Sao+xIT+G_info N1:resto	494	11,01	44,889	1,9169
ra_BasketballDrill_qp27	2	N0:Inter N1:resto	494	16,99	29,073	1,2415
ra_BasketballDrill_qp27	2	N0:Intra N1:resto	494	21,61	22,865	0,9764
ra_BasketballDrill_qp27	2	N0:Intra+Inter N1:resto	494	16,6	29,763	1,2709
ra_BasketballDrill_qp27	3	N0: Inter N1: Sao+xIT+parser N2: resto	494	12,83	38,504	1,6442
ra_BasketballDrill_qp27	3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	494	12,37	39,929	1,7051
ra_BasketballDrill_qp27	3	N0:Inter+Sao+qpGen N1:Intra+xIT N2:resto	494	16,99	29,079	1,2418
ra_BasketballDrill_qp27	3	N0:Inter+xIT N1:source+display+merger N2:resto	494	20,51	24,092	1,0288
ra_BasketballDrill_qp27	3	N0:Inter+Intra N1: qpGen+Sao N2:resto	494	16,77	29,461	1,2581

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
ld_BasketballDrill_qp32	1	Predeterminado	493	18,835	23,175	1,000
ld_BasketballDrill_qp32	2	Inter+Sao+xIT+G_Info	493	9,317	52,914	2,283
ld_BasketballDrill_qp32	2	Source+Inter+Qp_Gen+Sao+xIT	493	9,298	53,022	2,288
ld_BasketballDrill_qp32	2	Source+Inter+Intra+QpGen+Sao	493	9,302	52,999	2,287
ld_BasketballDrill_qp32	2	Source+Parser+Inter+Sao	493	9,273	53,165	2,294
ld_BasketballDrill_qp32	2	Inter	493	12,680	38,880	1,678
ld_BasketballDrill_qp32	2	Intra	493	19,286	25,563	1,103
ld_BasketballDrill_qp32	2	Inter+Intra	493	13,357	36,909	1,593
ld_BasketballDrill_qp32	3	N0: Inter N1: Sao+xIT+parser N2: resto	493	10,930	45,105	1,946
ld_BasketballDrill_qp32	3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	493	11,777	41,861	1,806
ld_BasketballDrill_qp32	3	N0:Inter+Sao+qpGen N1:Intra+xIT N2:resto	493	13,110	37,605	1,623
ld_BasketballDrill_qp32	3	N0:Inter+xIT N1: source+display+merger N2:resto	493	9,896	49,818	2,150
ld_BasketballDrill_qp32	3	N0:Inter+Intra N1: qpGen+Sao N2:resto	493	9,209	53,535	2,310

Sequence	Cores	.xcf	images	time(s)	FPS	Gain
ld_RaceHorses_qp27	1	Predeterminado	297	17,540	16,933	1,000
ld_RaceHorses_qp27	2	Parser+Inter+Sao	297	9,490	31,296	1,848
ld_RaceHorses_qp27	2	Source+display+Parser+Inter+Sao	297	9,652	30,771	1,817
ld_RaceHorses_qp27	2	Inter+Intra+Sao+Select	297	9,742	30,487	1,800
ld_RaceHorses_qp27	2	Inter+QpGen+Sao+xIT	297	9,539	31,135	1,839
ld_RaceHorses_qp27	2	Inter	297	13,570	21,887	1,293
ld_RaceHorses_qp27	2	Intra	297	16,198	18,336	1,083
ld_RaceHorses_qp27	2	Inter+Intra	297	13,182	22,531	1,331
ld_RaceHorses_qp27	3	N0: Inter N1: Sao+xIT+parser N2: resto	297	8,478	35,032	2,069
ld_RaceHorses_qp27	3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	297	9,075	32,727	1,933
ld_RaceHorses_qp27	3	N0:Inter+Sao+qpGen N1:Intra+xIT N2:resto	297	9,718	30,562	1,805
ld_RaceHorses_qp27	3	N0:Inter+xIT N1: source+display+merger N2:resto	297	10,184	29,163	1,722
ld_RaceHorses_qp27	3	N0:Inter+Intra N1: qpGen+Sao N2:resto	297	12,135	24,475	1,445
ra_RaceHorses_qp27	1	Predeterminado	296	15,673	18,885	1,000
ra_RaceHorses_qp27	2	Parser+Inter+Sao	296	8,693	34,05	1,803
ra_RaceHorses_qp27	2	Inter+Intra+Select+Sao	296	8,794	33,659	1,782
ra_RaceHorses_qp27	2	Parser+Inter+Sao+QpGen	296	8,684	34,228	1,812
ra_RaceHorses_qp27	2	Inter+Source+Parser+Sao	296	8,67	34,141	1,808
ra_RaceHorses_qp27	2	Inter	296	12,535	23,614	1,250
ra_RaceHorses_qp27	2	Intra	296	17,029	17,382	0,920
ra_RaceHorses_qp27	2	Inter+Intra	296	12,457	23,762	1,258
ra_RaceHorses_qp27	3	N0: Inter N1: Sao+xIT+parser N2: resto	296	10,391	28,486	1,508
ra_RaceHorses_qp27	3	N0: Inter N1: DBFilter+xIT+qpGen N2: resto	296	10,984	26,948	1,427
ra_RaceHorses_qp27	3	N0:Inter+Sao+qpGen N1:Intra+xIT N2:resto	296	12,653	23,394	1,239
ra_RaceHorses_qp27	3	N0:Inter+xIT N1: source+display+merger N2:resto	296	15,362	19,268	1,020
ra_RaceHorses_qp27	3	N0:Inter+Intra N1: qpGen+Sao N2:resto	296	12,573	23,543	1,247